Large-Scale Open Self-Organising Systems

Managing Complexity with Hierarchies, Monitoring, Adaptation, and Principled Design

Jan-Philipp Steghöfer

Dissertation zur Erlangung des Doktorgrades Dr. rer. nat.
der Fakultät für Angewandte Informatik
der Universität Augsburg, 2014

Erstgutachter:	Prof. Dr. Wolfgang Reif
Zweitgutachter:	Prof. Dr. Jörg Hähner
Tag der mündlichen Prüfung:	06. Februar 2014

Zusammenfassung

Sehr große Systeme—d.h., solche, in denen mehrere tausend unabhängige Komponenten miteinander interagieren und zusammenarbeiten—werden immer wichtiger in missions-kritischen Umgebungen. Ein prominentes Beispiel hierfür sind Energiemanagementsysteme, die durch die starke Verbreitung dezentraler Energieerzeuger auf Basis erneuerbarer Energienen einen enormen Größen- und Komplexitätszuwachs erfahren haben. Weitere Beispiele sind Zivilschutz und Katastrophenmanagementsysteme sowie globale Logistiksysteme. In solchen Systemen ist die bisherige, zentralisierte Kontrolle nicht mehr oder nur noch stark eingeschränkt in der Lage, die anfallenden Daten zu verarbeiten und rechtzeitig Kontrollentscheidungen zu treffen.

Der Schlüssel, um mit dieser Komplexität umzugehen, ist die Erhöhung der Autonomie der Systeme, also der Fähigkeit der einzelnen Komponenten, eigene Entscheidungen auf Basis der lokal verfügbaren Informationen und auf Basis von Koordination mit anderen Komponenten im System zu treffen. Eine vollständige Dezentralisierung bedeutet allerdings, dass es unmöglich wird, optimale Entscheidungen zu treffen, da die eingeschränkte Information, die bei dem einzelnen Entscheider vorliegen, dazu führt, dass übergeordnete Themen unter Umständen ignoriert werden.

Im Rahmen dieser Dissertation wird daher vorgeschlagen, hierarchische Systemstrukturen einzusetzen, in denen die einzelne Komponente—im folgenden auch Agent genannt—regionales Wissen über die Umgebung besitzt und in der Lage ist, Entscheidungen an übergeordnete Instanzen abzutreten wenn die eigene Informationslage ungenügend ist. Zu diesem Zweck wird ein *hierarchisches Selbstorganisationsverfahren* eingeführt, das hierarchische Strukturen etabliert, in denen Entscheidungen zeitnah und mit hoher Qualität getreffen werden können. Weiterhin wird eine *Infrastruktur zur Systembeobachtung* entwickelt, die auf Basis der Hierarchie Fehlverhalten des Systems entdecken kann. Diese Infrastruktur kann automatisch per Modelltransformation aus den zuvor erhobenen Anforderungen abgeleitet werden. In diesen wird das korrekte Verhalten des Systems spezifiziert, indem ein Verhaltenskorridor angegeben wird, bestehend aus harten Constraints—also Nebenbedingungen—und aus weichen Constraints, die wiederum mit Hilfe von *Constraint Relationships* ausgedrückt werden und optimales Verhalten angeben. Diese Infrastruktur wird gekoppelt mit Kontrollverfahren, die das System zur Laufzeit umkonfigurieren und adaptieren, indem sie Constraint Satisfaction Probleme lösen, die auf den selben Nebenbedingungen basieren wie die Verhaltenskorridore. *Modellsynthese und -abstraktion* propagieren diese Informationen und Kontrollentscheidungen dabei in der Hierarchie, so dass der richtige Entscheidungsträger immer die notwendigen Daten vorliegen hat. Ein agentenorientierter Software-Entwicklungsprozess erlaubt es schließlich, offene selbstorganisierende Multiagentensysteme in einer agilen, iterativ-inkrementellen Art und Weise zu erstellen, bei der die wichtigen und einmaligen Aspekte dieser Systemklasse berücksichtigt werden und entsprechende Hilfsmittel und Richtlinien zur Verfügung stellt.

Executive Summary

Systems of a very large scale—including several thousand independent components interacting and working together—become increasingly ubiquitous in mission-critical operations. A prominent example for this development are power management systems that have grown tremendously in size and complexity with the increased installation of distributed energy resources such as small solar installations and biogas plants. Other examples include civil protection and disaster management systems as well as planet-wide logistics systems. Centralised control in such systems is unable to process the amount of data that is produced and to make timely control decisions. The key to handling the complexity is thus increasing their autonomy, i.e., the ability of the individual component to make decisions on its own, based on its locally available information and in coordination with other components in the system. Full decentralisation, however, means that it becomes impossible to make optimal decisions since the limited knowledge of the decision maker forces it to ignore over-arching issues.

This thesis thus proposes to use hierarchical system structures in which agents have regional knowledge and are able to delegate decisions to superiors if their information is insufficient. For this purpose, it introduces methods for *hierarchical self-organisation* that create a hierarchical structure that is suitable to make timely, yet good decisions. It further details a *monitoring infrastructure* for hierarchically structured systems that can be automatically transformed from system requirements models and allows to detect misbehaviour in a hierarchical system. For this purpose, the correct behaviour of the system is defined in the same requirements, using hard constraints that define a corridor of correct behaviour and soft constraints—specified with *constraint relationships*—that define optimal behaviour. This monitoring infrastructure is coupled to controllers that can reconfigure a hierarchical system by solving constraint satisfaction problems—based on the same constraints as the monitoring infrastructure—and use *model synthesis and abstraction* to propagate information and control decisions through a hierarchical system. Finally, an *agent-oriented software engineering process* allows the development of open self-organising multi-agent systems in an agile, iterative-incremental way by incorporating important aspects of these systems into the process and providing important guidelines. The contributions are briefly summarised in the following.

Hierarchical Self-organisation

A hierarchical organisation of agents can be construed as a system of systems (SoS). Such structures allow operational and managerial independence of independent sub-systems that fulfil a function in their own right. An SoS A can be locally managed and becomes part of a larger system B that, again, is locally managed. However, the complexity within the system of system A is hidden from this larger system B since B interacts with a clearly defined interface and regards the whole of A as a single entity. Instead of controlling the individual components within A, the system B thus only has to control a single system, reducing the complexity and allowing A to locally manage the components without outside interference. Figure S.1 shows a hierarchical system structure.

This paradigm can be used with pre-defined SoS but can also be exploited in situations in which the SoS can be changed and re-arranged. This can be advantageous if the environment is volatile and the structure can react to this volatility by adapting itself and thus robustly provide the overall system goal. For this purpose, a hierarchical self-organisation algorithm called HiSPADA has been developed (cf. Chapter 4). It adapts a hierarchy based on application-specific criteria and allows the creation of purposeful, dynamic systems of systems. HiSPADA is based on a set partitioning algorithm which it

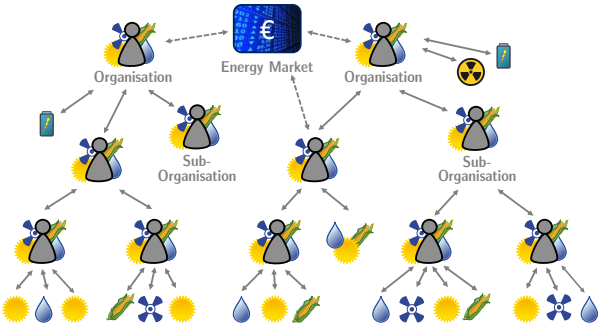


Figure S.1: A hierarchical system structure of power plants and AVPPs, organised as a tree. Higher-level agents observe and control their subordinates and in turn are observed and controlled by their superiors. This reduces the individual expenditure for these tasks since the number of controlled agents stays relatively low while providing the possibility to escalate important control decisions and aggregate information at higher levels.

uses to create sets of agents that are then represented by an intermediary. This intermediary provides the aforementioned interface to other systems and assumes control over the other agents in the SoS. In case of the power management example, this intermediary is an AVPP—an *A*utonomous *V*irtual *P*ower *P*lant.

HiSPADA is triggered by a violation of constraints, i.e., a violation of a condition that defines correct behaviour of the system. For instance, the time the intermediary requires to make its control decisions can be crucial in systems with strictly defined time constraints. Power plants, e.g., require a new schedule, determining their output at a given point in time, every 15 minutes. If an intermediary in such a system is not able to calculate the schedules in time, the purpose of the SoS is not fulfilled. In such a case, a constraint is violated and HiSPADA reorganises the hierarchy to re-structure the system in a way that reduces the scheduling time.

Constraint Relationships

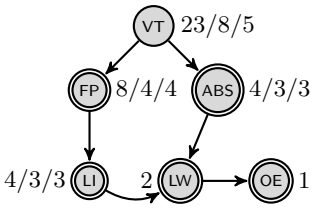
While the violation of the timing constraint HiSPADA reacts to is a very clear cut case, many other problems in large-scale open multi-agent systems can not be as easily defined. The scheduling problem itself, e.g., is very complex and requires models that describe the power plants physical constraints as well as economical and technical preferences. While it might, e.g., be physically possible to power-up a generator and power it down again within a short period of time, this might not be desirable economically and put strain on the machine that makes it technically undesirable as well. Such behaviour should not, however, be strictly forbidden, since it can be necessary to use a generator this way to stabilise the power grid.

To specify such circumstances, a formalism is required that not only allows to define soft constraints but also their relationship to each other. If several soft constraints are defined and different solutions to the problem are possible, this relationship indicates which constraint can be violated with less repercussions than others. For instance, it might be preferable to drop a soft constraint that limits the output of a generator to a small area around its "economic sweet spot"—i.e., the power output that has the highest efficiency—than to drop a soft constraint that prohibits drastic changes in power output. If we call the first constraint *ecs* for "economic sweet spot" and the second one *ndc* for "no drastic changes" we can thus define the relationship $ecs \prec ndc$, indicating that *ndc* is more important than *ecs*.

These constraint relationships (cf. Chapter 5) can be used to specify complex preference networks on constraints. These networks can then be translated into weights for the individual constraints according to different dominance properties. These dominance properties indicate how much more important constraints are than their predecessors in the partial order implied by the relationships. Single predecessor dominance indicates that violating a less important constraint rather than an important one should be considered better—ceteris paribus. When constraints are more important than a whole set of constraints that are explicitly denoted to be less important, direct predecessor dominance is used. Finally, if a constraint is more important than all directly and indirectly related constraints, transitive predecessor dominance is used. Figure S.2 depicts how the weights of constraints differ for different dominance properties.

Behaviour Monitoring and Observer-Synthesis

Since correct behaviour is specified with constraints—which in turn are predicates on the state of the system—an infrastructure has to be in place in the system to observe the system state, evaluate the



A weighted constraint relationship graph.

	Weight
$t_X^{(1)}$	8
$t_X^{(2)}$	13
$t_X^{(3)}$	**7**

SPD semantics.

	Weight
$t_X^{(1)}$	11
$t_X^{(2)}$	13
$t_X^{(3)}$	**10**

DPD semantics.

	Weight
$t_X^{(1)}$	27
$t_X^{(2)}$	**19**
$t_X^{(3)}$	25

TPD semantics.

Figure S.2: A constraint relationship graph, annotated with the weights for different dominance properties and the toal weights for three different solutions $t_X^{(1)}$ to $t_X^{(3)}$. Each solution violates a different set of constraints and thus implies a different accumulated weight or penalty. Depending on the dominance property, the best solution can differ.

constraints, and react to their violation (cf. Chapter 6). This infrastructure is based on the Observer/-Controller architectural pattern and can be automatically transformed from the requirements documents (cf. Chapter 7). In particular, the requirements documents contain the constraint specifications. Requirements and thus constraints are directly assigned to the agents responsible for their fulfilment and therefore for the observation of the constraints. The generic Observer/Controller model depicted in Figure S.3 is used in the transformation.

The transformation process supports hard constraints as well as soft constraints. While a violation of a hard constraint causes a reconfiguration of the system immediately, a soft constraint can be violated without triggering such drastic measures. Instead, a controller can try to optimise single parameters to fulfil the constraint again. This way, soft constraints also specify qualitative preferences on the states of the system when observed during runtime.

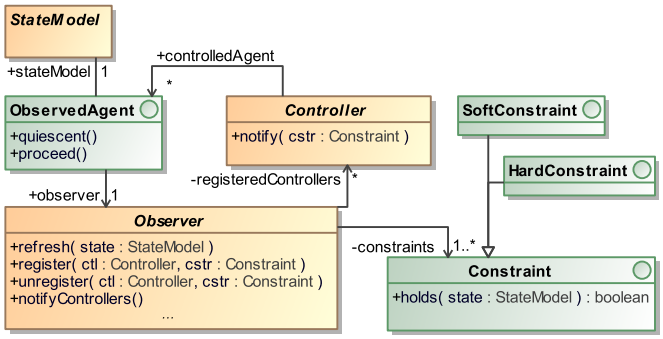


Figure S.3: The simplified generic Observer/Controller model used in the transformation, specified as a UML class diagram. The agents have to implement the `ObservedAgent` interface and call the `Observer.refresh()` method whenever their state changes. The observer then checks whether the constraints hold and informs the controllers if any violation is detected.

Hierarchical observation and control is supported as well. In a hierarchy, the observers of the different levels are connected to each other, mimicking the structure created by agents and intermediaries. Whenever the state model of a low-level agent changes, these changes are propagated to its intermediary's observer. It then synthesises the state models to gain a regional view of the agents it controls—i.e., the SoS it represents—and evaluates constraints on this synthesised model. An intermediary in turn submits its own state model to its superior. This technique ensures that systems are separated from each other, maintaining the systems of systems spirit.

Specification of Adaptation Processes as Constraint Satisfaction Problems

Most of the constraints used to specify the correct behaviour of the system can also be used to specify the adaptation processes that reconfigure the system so that it again adheres to the correct behaviour. More specifically, it is possible to use the constraints as part of constraint satisfaction models that describe valid solutions to, e.g., the scheduling problem for power plants, or the problem of creating purposeful hierarchical structures (cf. Chapter 8 and Chapter 9).

In some cases, however, the constraints observed at runtime are not suitable to describe a constraint satisfaction problem specifying the adaptation process. The constraints on the runtime of the scheduling process that drive HiSPADA, e.g., can only be evaluated at runtime when the actual times can be measured. An adaptation process that changes the system structure, on the other hand, has to make

assumptions about these runtimes based on other information available. For this example, the number of controllable power plants is the decisive factor that determines how long the scheduling process will take. Therefore, a constraint limiting the number of agents per partition or equalising the number of agents per partition can be used.

Specifying adaptation processes this way can be useful in rapid prototyping and in making sure that the requirements for the adaptation processes are complete. During system design, using a constraint solver to test the specification of the process reveals omissions or whether the specification is over-constrained. At runtime, the problems can, however, be solved with heuristics, such as the HiSPADA algorithm.

Synthesis and Abstraction of CSP Models in Hierarchical Systems

The power plant scheduling problem is an example of a process that requires information from flowing up the hierarchy while commands—i.e., the schedules—are flowing down. A physical power plant provides its model determining its physical limitations and economical and technical preferences to its intermediary—i.e., its AVPP. The AVPP collects and synthesises the models into a solution model (cf. Chapter 10). The solution model defines a constraint satisfaction problem that includes the constraints of all power plants and thus allows the AVPP to find a valid solution. The same technique is used to create the synthesised state models mentioned before.

The AVPP in turn is controlled by its superior which creates a schedule for all AVPPs and power plants it controls (cf. Figure S.1). Propagating the synthesised model upwards would lead to an increase in complexity for the model that has to be solved on a higher level of the hierarchy and increase the likelihood of creating an over-constrained problem, i.e., a problem for which no solution that adheres to all hard constraints. Instead, an AVPP has to abstract the synthesised model so that it provides a simplified model to its AVPP while maintaining high fidelity and ensuring that its underlying power plants can deal with the schedule provided from above (cf. Chapter 11).

Agent-oriented Software Engineering Process

The techniques introduced so far provide algorithmic and conceptual solutions to problems that occur in large-scale open self-organising multi-agent systems. The agent-oriented software engineering process PosoMAS provides guidance on how to integrate these ideas in the development process of such complex systems (cf. Part V). It is based on the relatively lean, agile, and iterative-incremental Open Unified Process and adapts and extends it by using custom practices that provide tasks, work products, references, and roles that are tailor-made for the system class. Tasks are bundled in activities that are structured within the phases of the process. Due to its modular structure, PosoMAS can be customised and itself be extended and adapted, e.g., by incorporating practices from other agent-oriented software engineering processes that focus on particular agent architectures such as BDI. The process is validated by applying it to two case studies, namely the autonomous power management system and an emergency response system.

Contents

List of Figures

List of Tables

Part I

Large-Scale Open Self-Organising Systems

Introduction to the system class, identification of the most important properties, challenges, and of the scope of the dissertation. Establishes two case studies used to exemplify the findings and evaluate the approaches and algorithms.

Chapter 1

Characteristics of Large-Scale Open Self-Organising Systems

Summary. Introduces the system class this thesis is concerned with and the particular challenges tackled in this document, especially the scale of large infrastructure systems, the robustness requirements of mission-critical infrastructure, command and control in such systems, as well as software engineering for large-scale open self-organising systems. The contributions of this thesis are listed and an overview of the structure of the dissertation is given.

Modern distributed computing systems are rapidly developing towards a complexity that we are just now starting to understand in all its potential and intricacy. This kind of complexity is, however, organised (Weaver, 1948), meaning that "a sizeable number of factors [...] are interrelated into an organic whole". This organic whole has emergent properties that are desirable and that system engineers want to exploit in their endeavour to create robust, adaptive, and long-living systems. The systematic design of "interrelations" in organised complex systems is the major research question in the branch of computer science that deals with complex adaptive systems or emergent and self-organising systems.

This thesis is concerned with the reduction of this complexity, both at design time and at runtime. For this purpose, a number of techniques and approaches are proposed that allow to design large-scale open self-organising systems, enable autonomous structuration at runtime, allow their effective observation, and the change of their reconfiguration in reaction to changes in the system and the environment. The main drivers of the development of these techniques are the sheer size of the systems, the importance they will achieve in the future and the corresponding requirements on their robustness, the necessity to make control decisions with a high quality, and the need for design processes that can deal with the challenges of the system class. The main solution approach is the use of systems of systems and the compartmentalisation such a view on a large-scalse system provides.

1.1 Scale of Future Infrastructures

The number of independent sub-systems—henceforth called *agents*—in systems providing vital infrastructural services is ever-increasing. A prominent example for such a very large-scale mission-critical infrastructure is the power grid. According to EnergyMap.info[1] more than 1.4 million power producers where online in October 2013 in Germany. This enormous system is strongly connected by the European power network UCTE[2] and interacts with more than 40 million households[3] and countless industrial, commercial, and public outfits that consume this power. This system—apart from its massive scale and the multitude of interdependencies it has internally, with the environment, and with the regulating bodies—is *open*, since it is possible to quickly install new generators and consumers, and *heterogeneous*, since it is practically unlimited in the kinds of generators and consumers.

[1] `http://energymap.info/energieregionen/DE/105.html`, requested October 20, 2013.

[2] From the former Union for the Co-ordination of Transmission of Electricity, whose functions are now fulfilled by the ENTSO-E, European Network of Transmission System Operators for Electricity.

[3] Website of the Bundeszentrale für politische Bildung, `http://www.bpb.de/nachschlagen/zahlen-und-fakten/soziale-situation-in-deutschland/61584/bevoelkerung-und-haushalte`, requested August 1, 2013

So far, such systems have been controlled by using centralised optimisation and limited observation. However, the size of these systems increases: only ten years ago, a meagre 100000 power producers were connected to the power grid. Likewise, the size of other critical infrastructure such as civil protection and disaster management systems, planet-wide logistics systems, and military command and control networks is reaching ever new records. This makes it more difficult to collect information centrally since the communication effort increases, to make timely decisions based on standard control algorithms, and to distribute and decompose these decisions to the individual agents in the field. One approach to deal with these issues is to increase the *autonomy* of the individual agent, thus allowing it to make independent decisions based on local knowledge. However, fully decentralising decisions makes it impossible to arrive at optimal solutions and difficult to observe the system, especially when it is open and heterogeneous.

To counter these disadvantages, a viable compromise between decentralisation and observability and optimality has to be found. Appropriate system structures and compartmentalisation of the system are the keys to finding this compromise. This can be achieved by structuring the system in a hierarchical fashion, akin to *systems of systems* (Sage and Cuppan, 2001) where parts of the system form a closed system in itself that interacts with other such systems. It is often not useful to provide this structure at design time but rather let the system find the structure itself in a process of *self-organisation* at runtime. This ensures that the structure and the current operating environment fit and that changes in the environment and within the system can be reflected by an adapted structure. Architectures and algorithms to achieve such a compartmentalisation are presented in Part II.

1.2 Robustness of Self-Organising Mission-Critical Infrastructure

The foundation of a robust system is the ability to adapt to changing environmental circumstances and to changes and faults in the system itself. Self-organising systems provide an excellent foundations to put such requirements into practice. They are inherently adaptive and able to find the adaptations necessary without external guidance (Heylighen et al., 2001). In addition, a system's emergent behaviour, stemming from the interactions between the agents, needs to be controlled at runtime. Agent autonomy, openness, and heterogeneity demand that the results of interactions are checked against their potential to harm the system or jeopardise the system goals. Thus, self-organising systems must be able to detect changes and identify erroneous behaviour or system states that could lead to faults or hazards. Such faculties are especially important in mission-critical infrastructure such as the power grid where failures can cause damages both financial and physical and which a great number of people rely on implicitly. As such systems are often deployed in highly volatile environments, external influences can also endanger the system's functionality and safety. The robustness of the system therefore crucially depends on its ability to detect and adapt to situations outside of its original specification.

Nafz et al. (2011) developed the *R*estore *I*nvariant *A*pproach (RIA) that allows to check whether a system is within a behavioural corridor at runtime. The corridor is specified with constraints on the allowable system states. Is one of the constraints—and thus, the corridor—violated, the system detects this violation and triggers an adaptation that *restores* the system back into a state in which the *invariant* is upheld. While this concept has proven very powerful both for the specification of self-organising systems (Nafz et al., 2010) and for use during runtime (Nafz et al., 2009a), it did not yet account for preferences that define a "soft corridor" in which the system behaves optimally. In addition, the infrastructure required to monitor the constraints had to be constructed by hand and adapted for each individual system manually. The techniques outlined in Part III address both problems by providing a technique for the constraint-based specification of preferences and for the automatic synthesis of an observer infrastructure from the system requirements.

1.3 Command and Control in Large-Scale Self-Organising Systems

The observation infrastructure can detect unwanted behaviour, but once detected, a reaction has to be found that counteracts it. Appropriate measures need to be taken to automatically stabilise the system even in situations in which it approaches critical conditions. The constraints used to specify the corridor of correct behaviour can be reused for the specification of the control algorithm (Nafz et al., 2009b) by using them as part of constraint satisfaction or optimisation models. However, such models are mainly applicable in centralised approaches and even decentralised solution approaches rely on centralised information (Yokoo, 2001).

To make them usable in hierarchical systems, new techniques are necessary that allow the local aggregation of information to formulate models in different compartments or regions of the system. Following the systems of systems approaches, these regions autonomously solve the command and control problems based on the knowledge about the constituents of the region. To accommodate the hierarchical composition of the system, the models used in the solution need to be propagated to higher hierarchy layers while limiting the complexity these higher layers have to deal with. Thus, models need to be abstracted in a way that allows meaningful control decisions on the higher level that can be used on the lower levels with as little information as possible. These issues are discussed in more detail Part IV.

1.4 Software and System Design for Open Self-Organising Systems

The concepts, techniques and algorithms developed to deal with the complexity of large-scale open self-organising systems need to be available to software and systems engineers in an accessible form. In particular, the integration of the techniques into a software engineering process enables a principled approach to the construction of such systems. Agent-oriented software engineering provides a number of methodologies that incorporate principles of multi-agent systems, self-organisation, and respective system structures (see, e.g., Padgham and Winikoff, 2005; DeLoach and Garcia-Ojeda, 2010; Cossentino et al., 2010), but these processes are rather rigid, provide little help with specific aspects of open systems, or prescribe a certain way of modelling the agents or the system.

Traditional software engineering methods (such as the Unified process, see, e.g., Kruchten, 2004) on the other hand are very general and thus provide no guidance for agent-based systems. Their strengths are in their pervasiveness and their excellent documentation. An ideal engineering methodology for the system class considered in this thesis combines both worlds, providing a well-known and proven development life-cycle, good documentation, and adaptations to open self-organising system without making too many assumptions about the agent architecture used, the tools employed, or the final system structure. Furthermore, the method content is provided in a way that makes it possible to combine it with different process frameworks and allows the resulting process to be tailored for the project at hand. The discussion in Part V addresses these issues and proposes solutions to these challenges.

1.5 Trustworthy by Design—Contributions of this Thesis

At design time, measures can be taken to ensure the trustworthiness of a system by regarding different facets of trustworthiness, such as reliability, usability, functional correctness, and safety during the engineering process. Tools such as formal verification, the use of proven techniques such as design patterns, and the thorough and diligent design of algorithms and incentive systems support the engineer in this endeavour. In some domains, such as automotive systems, dedicated certification processes exist to ensure the trustworthiness of the systems. In general, this kind of trust is an external property of a system, based on the perception of the system by outsiders (Steghöfer et al., 2010). The goal of this thesis is to provide tools that ensure that a system is perceived as trustworthy through the systematic use of techniques that reduce complexity and allow the system designers' to effectively deal with the challenges outlined above. The concrete contributions are exemplified with a decentralised power management system as the main case study and an emergency response system to outline additional aspects. Both are introduced in Chapter 2. The contributions of the thesis are as follows:

Hierarchical Self-organisation The foundation for observation and control in large-scale open self-organising systems is a hierarchical self-organisation approach based on the principles and architectures outlined in Chapter 3. It uses a set-partitioning algorithm to create hierarchies driven by application-specific constraints. This allows the system to self-organise a hierarchical system structure in which observation and control are both efficient and localised. The algorithm and evaluation results are presented in Chapter 4.

Constraint Relationships The effective observation of behaviour in large-scale self-organising systems requires not only strict behavioural boundaries, expressed as constraints on the system state, but also the observation of preferences and possibilities to react to their violation. In addition, the formulation of individual preferences in decision models allows the optimisation with regard to these preferences while at the same time achieving over-arching goals. The foundation for both applications

of preferences are constraint relationships, introduced in Chapter 5 that allow the formalisation of a "more important than"-relationship between constraints.

Behaviour Monitoring and Observer-Synthesis The observation of the system's behaviour and state allows reactions to misbehaviour and allows the prevention of potentially hazardous decisions. The desired behaviour is specified as part of the requirements elicitation activities as outlined in Chapter 6. The same chapter details a flexible and capable monitoring infrastructure that incorporates the observation of strict behavioural corridors and of preferences in systems with flat and hierarchical system structures. Since all necessary information to adapt this infrastructure to a specific system is collected at design time, it can be synthesised from the requirements models as explicated in Chapter 7.

Specification of Adaptation Processes as Constraint Satisfaction Problems The basis for adaptation processes and control decisions as forms of feedback loops are laid in Chapter 8. The same way constraints defined on the system state can be used to monitor correct behaviour at runtime, they can be used to specify correct solutions for these adaptation and control decisions. The way decisions and adaptations in the case studies can be expressed as constraint satisfaction and optimisation problems, how preferences are combined with these problems, and how uncertainty can be incorporated into the decisions is described in Chapter 9.

Synthesis and Abstraction of CSP Models in Hierarchical Systems Hierarchical systems allow the regio-central solution of adaptation and control decisions as a compromise between scalability benefits of decentralised solutions and the optimality benefits of centralised ones. To use constraint satisfaction techniques in these systems, it is necessary to allow the synthesis of a combined model from simpler, individual models as explained in Chapter 10 and the abstraction of such synthesised models on higher levels of the hierarchy as explicated in Chapter 11.

Agent-oriented Software Engineering Methodology The final building block is to provide a standardised software engineering methodology that allows the principled development of large-scale open self-organising multi-agent systems. In comparison to most existing agent-oriented methodologies, outlined in Chapter 12, the *P*rocess for *o*pen *s*elf-*o*rganising *M*ulti-*A*gent *S*ystems—PosoMAS—makes only very few assumptions about the structure of the solution and does not prescribe the use of certain agent architecture or meta-models while providing the development team with activities, guidance, and a life-cycle to successively develop a system in the class considered in the thesis. While the methodology itself is described in Chapter 13, its application to the two case studies and its evaluation in comparison to existing agent-oriented software engineering methodologies is presented in Chapter 14.

The research contributions and the most important evaluation results are summarised in Chapter 15. Open research challenges and future directions are discussed in Chapter 16.

Publication of the Material of this Thesis

The approaches, concepts, and algorithms that form the contributions of this thesis have been published by the author in various peer-reviewed conferences, workshops, and journals. The most important publications are listed below and a complete list of publications of the author can be found in the back matter on page 269.

- (Steghöfer et al., 2010) and (Steghöfer and Reif, 2012)—giving an overview of the challenges on the trustworthiness of open heterogeneous multi-agent systems;
- (Steghöfer et al., 2013a)—describing the challenges and solution approaches for autonomous power management systems;
- (Steghöfer et al., 2013b)—detailing the hierarchical self-organisation algorithm;
- (Schiendorfer et al., 2013)—laying the formal foundations for constraint relationships;
- (Steghöfer et al., 2013c) and (Eberhardinger et al., 2013)—outlining the approach to behaviour monitoring and observer-synthesis;
- (Schiendorfer et al., 2014)—detailing model synthesis and abstraction in hierarchical systems;
- (Seebach et al., 2010) and (Sudeikat et al., 2012)—providing insights into software engineering for self-organising systems;

Chapter 2

Case Study: Decentralised Power Management

Summary. Power management systems of the future will be large-scale open self-organising systems that directly incorporate distributed energy resources into the scheduling processes, react much quicker to changes in power demand and production than today's, and will allow small groups of producers and consumers to participate in the power market. All this will lead to a more flexible, robust, and scalable power grid, at least if the technological challenges can be mastered. Agency of the power plants, trust as a measure of uncertainty, and self-organisation will be the keys to tackling the challenges at hand. This chapter introduces the running case study and outlines its most important characteristics.

It also briefly introduces an emergency response system, another large-scale open self-organising system used throughout this thesis with complementary characteristics.

Publication. The concepts outlined in this chapter have been published in (Steghöfer et al., 2013a).

To illustrate the techniques and algorithms developed in this thesis, two case studies will be used, with a strong focus on the first one, *autonomous power management systems*. The main part of this chapter is therefore devoted to this prospering field that has been discovered by computer scientist some years ago and has since become a catalyst for innovation and inventiveness, especially in the multi-agent systems community. Mainly associated with the term "smart grid", power management systems have become the centre of attention due to the massive changes that are currently taking place in this field, necessitated by the advent of distributed energy resources (DER) and the networking capabilities that more and more power generators and consumers possess. The shift towards electric mobility is also often cited as a driving factor of this revolution in the way power systems are managed. The main driver of innovation is the need to make power management autonomous and self-managing, based on the massive scale and connectedness that power management systems have achieved. In Section 2.1, the current state of power management systems and the future we envision for them will be introduced. The characteristics of these new kind of systems will then be discussed in Section 2.2, before outlining solution approaches to the changes that will occur Section 2.3.

The second case study, a *Self-organising Emergency Response System* will be introduced in Section 2.9 as an example of a large-scale open socio-technical system. In comparison to power management, involvement of the human is much more important here and the self-organisation inherent in this system is not the result of algorithmic intervention but of human decisiveness.

2.1 Power Management Systems—Now and Then

To illustrate the proposed transition towards a self-managing power system, we first outline the current organisation of power management systems and contrast it with the vision of the future system at the end of the transition. The description is based on the synchronous grid of Continental Europe, but structures are similar in other parts of the world.

Current power management systems: The current organisation of power management systems is depicted in Figure 2.1. Big power plants are controlled by electric utilities and other organisations in a flat hierarchy. For each of the big power plants, a schedule is created that postulates the output

of the power plant at a given time. Schedules are coarse-grained, providing target values in 15 minute intervals. Small power plants and especially DERs under the control of small cooperatives or individuals produce without external control and feed the power produced into the grid. This lack of control by the electric utilities—which are, after all, responsible for maintaining grid stability and balancing power consumption and production in balancing groups—is compensated by powerful controllable power plants. Current plans are to scale the controllable output further by installing more plants, especially flexible gas-powered ones that can be powered on within seconds. Very little measuring equipment is available in the field, so the actual grid status is unclear and predictions that guide energy production are based on intuition, weather forecasts, and historical analysis instead of current, reliable data. Nowadays, they are made by humans or SCADA (Supervisory Control and Data Acquisition) systems in the organisations. Additional power required to maintain the balance between production and consumption is provided by the operating reserve. The continental European grid, e.g., has 3 GW of idle power that can be activated within seconds. To be able to account for imbalances, the power grid is structured into balancing groups where utilities are responsible for equalisation of power consumption and production in each group (UCTE, 2009). Further, entrance requirements to power markets, such as the lower limit of 100 kW for contracts at the European Energy Exchange (EEX), exclude access for small organisations.

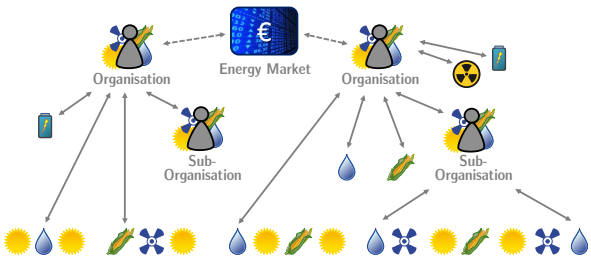


Figure 2.1: Current structure of power management systems: hierarchies within organisations are flat, some power plants are controlled by organisations directly, and only sufficiently large organisations participate in the power market.

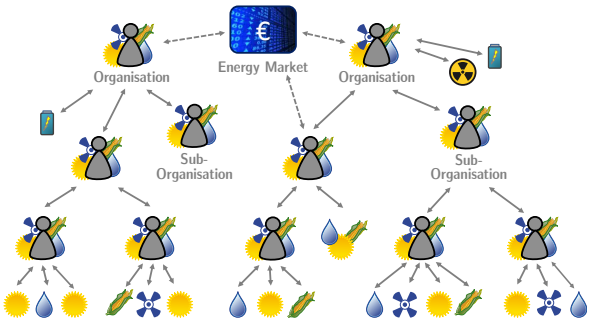


Figure 2.2: Hierarchical system structure of a future power management system: power plants are structured into SoS represented by AVPPs, decreasing the complexity of control and scheduling. AVPPs participate in the power market and can be part of other AVPPs.

Future power management systems: We envision a power grid in which even small power plants can be controlled or participate in a scheduling scheme. For this purpose, the system is structured into hierarchical systems of systems, as depicted in Figure 2.2. Networked measuring equipment allows to observe the grid status and make decisions based on current conditions. Power plants and consumers are networked, too, and predict future production or consumption. The quality of these predictions is known and can be used to derive possible scenarios for the future to guide scheduling decisions. Groups of power plants are represented by specialised SoS—*Autonomous Virtual Power Plants* (AVPPs)—that create schedules to cover a portion of the load and can flexibly access the power market. AVPPs can represent utilities, small organisations such as cooperatives of farmers that run biofuel plants, or simply a number of DERs that are controlled by the same organisation. Schedules are not only optimised for the economic benefit of large organisations but also take preferences of the individual participants into account. The behaviour of AVPPs can be guided by the controlling organisations. Finally, the operating reserve is provided by all controllable power plants in a decentralised scheme, superseding the need for expensive reserve power.

To make this vision—or a quite similar one—a reality, a number of fundamental changes have to occur. Many of them are legal and political, but some of them are technological. As computer scientists, we are mainly concerned with the latter kind, although an intensive dialogue between the different disciplines, politicians, companies, and consumers is of the essence. The solutions we propose in this paper and elsewhere are all technological and based on the notions of self-organisation and quantified uncertainty with trust values.

Several different approaches to re-structure power management systems have already been proposed. A key concept is the Virtual Power Plant (VPP), usually defined as groups of power plants that are in most cases controlled by a central entity (see, e.g., Lombardi et al., 2009). Sometimes, membership in

a VPP is restricted to certain types of power plants or power plants with predefined properties, such as dispersed generation units and micro-CHPs (combined heat and power units) in (Schulz et al., 2005) or DERs in (Bel et al., 2007). These approaches mainly focus on providing structures to integrate DERs into existing control schemes. Others concentrate on facilitating trading in power markets and distinguish commercial VPPs that participate in power markets and technical VPPs that provide services for the transmission net (Pudjianto et al., 2007). None of these approaches combine all the features that we are looking for in the power management system of the future. However, they give important insights into the organisation and functionality of such a system.

2.2 Power Management as a System of Systems

Power management systems are *systems of systems* (SoS). Intuitively, a system of system is a complex system that is itself composed of complex systems. There are five characteristics of SoS which are elaborated in Section 3.2: operational and managerial independence; geographical distribution; emergent behaviour; and evolutionary development.

Modern power management systems fulfil them all. Each power plant can be operated independently from the others. Although generators are coupled due to the shared power network and regulatory frameworks, economic operation of the individual plant is—to a certain extent—independent from others. Utilities and companies manage parts of the overall power system independently from each other. Geographical distribution is even increasing with the wide-spread installation of distributed energy resources (DERs) such as biofuel plants, solar plants, and wind farms. The relative stability in the network is an emergent behaviour. No single entity of the system can provide this stability in the face of load fluctuations, weather changes, and generator outages. The system as a whole, however, provides safeguards and semi-automatic control regimes.

Of course, the state of current, centralised, manually managed power management systems is the result of an evolutionary development process. Current proposals are directed at steering this evolution towards escalating autonomy and decentralisation. Such an evolution (see, e.g., Smathers and Akhil, 2000; Ramchurn et al., 2012) is necessary to deal with the growing number of generators, the increasing dependence on unreliable sources, and the increasing ability to control DERs. The vast scale of future power management systems and the necessity to deal with the self-interest of the individual operator of a power generator or power consumer will make it necessary to consider compartmentalisation, hierarchical structures, and decentralised problem solving. It is also necessary to accommodate existing organisational structures, i.e., to represent both the logical structure of utilities and their subsidiaries and the structure of the power grid which is divided into low-voltage, medium-voltage, and high- and highest-voltage segments.

Extensive studies have been performed on the technological areas that need to be covered by such a transformation and the impact and role the individual technologies will have. We will repeatedly refer to the acatech Future Energy Study (Appelrath et al., 2012) in this chapter, as it is very recent and comprehensive. It also stresses the importance of increased autonomy of power systems, especially in the context of virtual power plants. The transformation into decentralised, autonomous systems will not be a sudden shift but a gradual evolution that requires both tremendous technological and regulatory changes. It will be complemented by other developments, especially w.r.t. the way power consumers are integrated into the system.

2.3 Requirements and Solution Approaches for Decentralised Power Management

As computer scientists, our focus is on the software side of this process. Many of the contributions of this thesis have applications in power management systems and can support an evolution towards a scalable, autonomous system of systems. These changes can be deployed on a large scale or within individual organisations to make internal management systems more robust, flexible, and cost-effective. This chapter gives an overview of the requirements that have to be considered when designing software technology for decentralised power management systems. It also outlines techniques that have already been developed to deal with these requirements and a discussion of their validity and usefulness. Details can be found in the chapters referenced here. The techniques we propose should not be seen in isolation, but have to be complemented by other technologies and a shift in the legal and regulatory framework

(see, e.g., the results of the E-Energy project[1] as well as of several European and international initiatives such as the FP6 project INTEGRAL[2], the European Electricity Grid Initiative[3] and its associated projects such as DISCERN, or the UK's Autonomic Power Systems project).

Requirements for Autonomous, Decentralised Power Management

Some of the requirements of future power management systems have already been outlined. The following list gives an overview of the most important ones in the context of this thesis. For a more detailed look on the challenges and the environment these system will operate in, please refer to, e.g., Appelrath et al. (2012). The overall goal of the system will remain the balance of production and consumption of power and the economic and safe operation of the grid. With a greater autonomy, and decentralisation, however, additional possibilities come into reach, e.g., the relief of the current, centralised system of reserve power with a more economical distributed one in which all power generators participate (Anders et al., 2012a).

System scale: While in the past, system operators only had to deal with a manageable number of power plants which they had a detailed model of and could control directly and without limitations[4], the integration of distributed energy resources such as small biogas plants, combined heat and power units, or emergency generating unit will change how schedules are created and how primary, secondary, and tertiary power reserves are provided. Scheduling will have to take many more generators into account, often without knowing their exact control models but only based on threshold values about maximum and minimum output, as well as the generators rate of change.

With the increased number of generators also comes an increased availability of data. In principle, every system connected to the power grid can gather data about the current, local status and collect information such as network frequency and voltage levels. This information can be very valuable, e.g., when trying to avoid local voltage spikes that occur when a lot of power is input into the low-voltage band by photovoltaic installations and no consumption occurs at the same segment of the grid. The sheer volume of this data, however, can hardly be centrally processed and transformed into control actions in a short time.

Uncertainty about intermittent power production: Scale also affects the input of intermittent power generators such as solar and wind power plants into the system. The growing number of installations and their increased production can no longer be readily offset by power plants that can react quickly to drops or increases in the input. This introduces a level of uncertainty into the system that has, so far, been largely ignored. In future power management systems, the accuracy of the predictions will have to be factored in, however, since it determines the power that has to be scheduled (the so called *residual load* is the gap between the power consumption and the production by intermittent resources). If variations of the intermittent input are to be dealt with locally and in a distributed fashion by the power plants, reserves have to be scheduled accordingly as well.

Self-interested system participants: Each power generator operator is interested in making economical use of its generation capacity. For instance, small cooperatives of farmers that operate a biomass generator have an interest of running the generator at capacity whenever biomass is available. That these times may not coincide with the times in which the input is required is, at the moment, of little concern to them. To achieve stability in a future power management system it would be much better to run the generator when its input is helpful and required. In order to do this, the cooperative of farmers has to divulge the state of its generator as well as its control model. Appropriate financial or legislative incentives may be used to induce such a behaviour, but concerns of privacy and self-interest must be taken into consideration.

Observation and control: In a system of hundreds of thousands of system components, the flow of information and control has to be clearly defined. Responsibilities and authority has to be clearly

[1] `http://www.e-energy.de`

[2] `http://www.integral-eu.com`

[3] `http://www.smartgrids.eu/European-Electricity-Grid-Initiative`

[4] Munich's municipal utility ("Stadtwerke München"), one of the larger ones in Germany, controls about 50 generators, some of which are only producing heat for the district heating network or are intermittent wind or solar generators that can not be finely controlled. Overall, the utility has to create schedules that depend on power output for only 21 of these 50 generators. Source: `http://www.swm.de/dms/swm/dokumente/unternehmen/energieerzeugung/broschuere-erzeugungsanlagen.pdf`, visited Aug. 3rd, 2013.

regulated. As effective control of the system requires good knowledge of its status, the observation of relevant data about the system on all levels of analysis has to be ensured. The data has to be aggregated and collected at the appropriate locations in the system and used there to make control decisions that influence parts of the system. If these control decisions are made autonomously, the appropriate power to make these decisions has to be delegated by a legal entity such as a utility.

Longevity of the system: The next generation of power management systems will be gradually deployed and will be in use for the next decades. That means that all solutions that are designed now must be robust with regard to their future evolution and additions to the system, both on the hardware and on the software side.

Integration of controllable and stochastic consumers: So far, the discussion has mainly focused on power generators. However, power consumers can be a valuable asset in the creation of schedules and the stabilisation of the grid. The developments outlined for power generators also apply to power consumers: they are more often networked, can be potentially controlled from the outside, can provide data about the status of the grid, and can serve as storages (such as refrigerated warehouses or electric vehicles).

Solutions for Scalability and Openness

These unique requirements make innovative solutions necessary. The shift in power management systems can not simply be alleviated with the common approach of throwing a lot of computational power at the problem, since the requirements outlined above jeopardise the traditional approach of centralising control as it might not even be in the individual systems' interests to contribute to overall system goal and thus give control up.

We propose a solution to these requirements that introduces an updated system architecture with separate systems within a hierarchy each of which solves sub-goals and is incentivised by an internal or external framework to participate in reaching the over-arching system goals. This incentive-driven design returns some control over the system behaviour to the designer and can also be used to adapt the system to future changes in the environment at runtime. From a design perspective, a shift towards modularisation, compartmentalisation and emphasis on the interfaces between (partial) systems allows a separation of concerns between the design of the interaction of the system components and the components themselves.

While current limitations—such as missing measuring equipment—make the deployment even of the relatively simple features of an autonomous, decentralised power management system problematic, we believe that these features are invaluable contributions to the development of a truly smart grid, as e.g., called for in the staged development outlined for virtual power plants in the acatech study (Appelrath et al., 2012, Technology Area 12).

This thesis suggests a number of solutions for two of five feature sets that we propose as a contribution to future power management systems. Most importantly, it provides architectural solutions for many of the problems that occur in such large-scale open self-organising systems and a principled design methodology for their creation. The five feature sets are:

Agency of power plants and semi-autonomous creation of schedules: power plants and organisations work together to create power plant schedules that designate the power plants' output for specific points in time. At the same time, the power plants are able to act in their own interests and try to optimise their benefit while working within a framework that achieves the overall system goal. See Section 2.4 and Chapter 9.

Hierarchical self-organisation into autonomous virtual power plants: the system scales to large numbers of DERs due to a self-organised hierarchy that introduces intermediaries and allows compartmentalised schedule creation. The The introduction of these intermediaries is a key driver of the modularisation and compartmentalisation as well as the prerequisite to tackle scalability issues. See Section 2.5 and Chapter 4. These intermediaries are also natural places to observe and control the underlying power plants. See Chapter 6.

Trust to manage uncertainty: the system handles uncertain predictions from intermittent power generators and malicious participants using trust values and trust-based scenarios. See Section 2.6, Section 9.3, and (Anders et al., 2013a).

Trust-aware power markets: : power plants participate in power markets in their own economic interest. Prices and trade restrictions are based on trust and reputation of the market participants. See Section 2.7 and (Anders et al., 2013b).

Delegation of control with electronic institutions: organisations can control the behaviour of their subordinate power plants in a fine-grained way and revise rules based on changes in the organisational or regulatory conditions. See Section 2.8.

Each feature set provides direct benefits to organisations without having later features rolled out in the field as discussed in the respective sections. While this thesis is focused on power producers, mainly to avoid case distinctions when speaking about producers or consumers, controllable or stochastic power consumers can easily be integrated into the proposed schemes to increase the flexibility of consumption and enable a mixture of load-led and generation-led operation, e.g., by participating in scheduling and the reaction to frequency changes in the grid, be part of hierarchies, use trust-aware markets, and be governed by norms. Energy markets provide a natural interface between these power management applications.

The feature sets outlined above also contribute to the longevity of the system. The hierarchical system structure and the incorporation of self-interested agents into the system allow scaling even if the number of DERs keeps increasing at the current rate. The flexibility of the system w.r.t. its behaviour depending on organisational and regulatory conditions allows quick adaptation to changes in the political and business framework. The integration of techniques to deal with uncertainty gives the system the ability to adapt to a changing environment and strategically acting participants. Finally, the use of a principled design process and clear interfaces between the AVPPs and the power plants allows the integration of new types of generators and of consumers without the need of re-designing the systems as a whole.

Parts of the feature sets have already been implemented and validated in simulations, not only by us but by researchers around the globe. Results of the simulations we performed will be shown in the appropriate sections. The work conducted elsewhere is introduced there as well.

2.4 Agents as Representatives of Power Plants

The first step towards a more autonomous power management system is to enable power plants to proactively participate in the creation of schedules and in maintaining the stability of the grid. Each power plant can then optimise for individual criteria while maintaining global stability and quality. As many decisions require no human intervention, reactions can be faster. Schedules are based on better information due to distributed proposal creation and aggregation based on local weather forecasts, power plant status and load predictions. Together, this gives a strong incentive for small DER operators as well as larger utilities to participate in the scheme: they can better optimise internally and save money in the process while still meeting external goals. Agency of power plants, i.e., the ability to act autonomously in the environment, is the prerequisite for all other changes we propose.

This feature set requires that all power plants—especially DERs—and consumers have access to a communication network, making them accessible from the outside, and allowing the power plants to communicate their current status and predictions of their future output to other parties that create schedules. Merely networking generators to make them accessible from the outside is not enough, however, since the complexity and scale of the system requires them to act autonomously and proactively in the system instead of just providing reactive services that are activated by a central control. Of course, opening power generators like this incurs security and privacy challenges (see, e.g., (see, e.g., Ericsson, 2010; McDaniel and McLaughlin, 2009) and introduces requirements for communication languages and standards to enable semantic and syntactical understanding between the components (Appelrath et al., 2012, Technology Area 17).

The superordinate organisations are SoS as defined by (Sage and Cuppan, 2001) and are predefined by the electric utility responsible for a certain area. Each of these SoS is responsible for satisfying a specific part of the overall residual load—i.e., the load that has to be satisfied by controllable power plants—by creating a schedule for its subsystems that meets this demand. The schedule creation problem—a special kind of dynamic resource allocation (Anders et al., 2013b)—can be solved by a centralised approach (see, e.g., Heo et al., 2006; Zafra-Cabeza et al., 2008) within each organisation as before. Of course, the number of generators that has to be accommodated is much higher than before so that scalability problems are introduced that can be solved by introducing hierarchies in the system, either manually or automatically (see Section 2.5).

With agency of the power plants, the schedule is now based on the current status of controllable power plants, predictions by weather-dependent power plants, aand predictions of the future load, enabling more precise and cost-effective schedules. Especially the second point is a major difference to

current systems in which weather-dependent power production is predicted centrally or not at all. At this point, predictions are used directly in the scheduling process and no estimation of their accuracy is performed (see Section 2.6). Thus, a human operator still has to adapt predictions to current events, experiences made in the past, and so forth. The amount of data available about the power grid increases however, since in principle each power generator can act as a sensor and determine the local status, e.g., the network frequency and voltage bands. This information can be used to make locally effective decisions, e.g., to balance the voltage level in sections of the low-voltage network. Of course, this requires the data to be communicated in a timely fashion and an appropriate information processing infrastructure.

Schedules have to be adapted repeatedly as new data becomes available. Even though predictions for long time horizons are unreliable, they are required to identify if demand has to be covered on the market or surpluses can be sold. The scheduling of the residual load is based on models of the controllable power plants, their current status, and the status of the power grid. The models are formulated as constraint optimisation problems. Constraints describe the physical properties of the plants, such as minimal and maximal output, rate of change, as well as shutdown and warmup times. Another factor is the production cost of power, given as a function of the output. Status information includes current output and whether plants are on or in hot or cold standby. A constraint optimiser calculates the optimal output for each power plant so that the combination of outputs is as close to the predicted demand as possible and as cheap as possible.

Controllable power plants adopt this schedule and try to fulfil the assigned output as well as possible. To detect deviations from the target output, sealed and secure measuring equipment, similar to smart meters now deployed in households, has to be available at each power station. If these units report deviation from the schedule over a secure network, accounting systems can automatically reduce compensation.

2.5 Hierarchies of Autonomous Virtual Power Plants

To compensate the increased complexity of data management and schedule creation, we propose to introduce a hierarchy of systems of systems—composed of the individual power plants and intermediaries that represent groups of them—that self-organises to adapt to a changing environment. Grouping power plants allows to decompose the scheduling task and thus satisfy the power demand in a robust fashion.

The autonomous creation of power plant schedules is an NP-hard problem. Not only is a centralised solution computationally expensive, it also requires a lot of communication to propagate the information required to articulate the models. In the system described so far, the organisations acquire the necessary information and create the schedules. This leads to performance bottlenecks that become more severe as the system grows. At the same time, the system becomes more susceptible to errors since failures in communication and failures of individual agents could compromise the system. Thus, to increase scalability of the system, we propose a hierarchical system structure as depicted in Figure 2.2. Such a structure will also increase efficiency in coping with uncertainties and untrustworthy agents as well as complying with schedules, market contracts etc. Parts of the SoS hierarchy will be predefined to represent the existing structure of utilities, grid operators, or network topology. However, within that organisational structure, a self-organisation process can guide the adaptive creation of hierarchy levels and group power plants in Autonomous Virtual Power Plants (AVPPs). Each AVPP constitutes a system of systems. AVPPs reflect the structure of the power grid and can be located at different voltage levels. Internally, they consist of a mix of generators and energy sources and their main objective is to locally balance demand and supply.

An AVPP acts as an intermediary between the power plants and the organisation they are associated with. The intermediary thus assumes some of the responsibilities of the organisation, including the observation and control of the power plants. It thus has to be itself under the control of the organisation and fulfil the roles that have been delegated to it. In the first incarnation of this hierarchical systems, this delegation can be defined implicitly by the functionality implemented in the AVPP agents. More advanced versions of the system can, however, include a fine-grained delegation system based on norms (see Section 2.8). This allows explicitly defining the authorities of an AVPP and the regulations within which these authorities can be assumed. For instance, an organisation could grant an AVPP the right to interact with a power market, but only up to a certain monetary value or up to a certain amount of power.

The intermediary in turn observes and controls the behaviour of the power plants. That includes observing the quality of the predictions made and the adherence to the schedules. If the intermediary detects misbehaving agents, it has the liberty to address this misbehaviour, e.g., by reducing the share of the power scheduled for a power plant (see Section 2.6). This in turn can induce an agent to behave better—as long as its in the power of the agent to change its behaviour—to avoid monetary losses. This kind of observation and control is called *regio-central*, as it is at the same time regional for a limited group of agents (i.e., exactly those power plants controlled by the AVPP) and central since the intermediary collects the data and makes control decisions.

Intermediaries are located on the meso-level of the system. As depicted in Figure 2.3, the goals and the constraints on the meso-level differ from those defined on the micro-level of analysis that contains the power plants, and the macro-level that encompasses the entire system. The meso-level has to allow take the goals of the power plants into consideration and has to pursue its own goals at the same time. The emergent effect of the processes on the meso-level should then yield the overall functionality at the macro-level (see characteristic 4 of systems of systems as defined in Section 3.2). Thus, control approaches on the meso-level provide the overall functionality and deal with the complexity of the overall system. The hierarchies introduced into the system allow compartmentalisation and thus decrease the overall complexity by dividing the problem into smaller sub-problems.

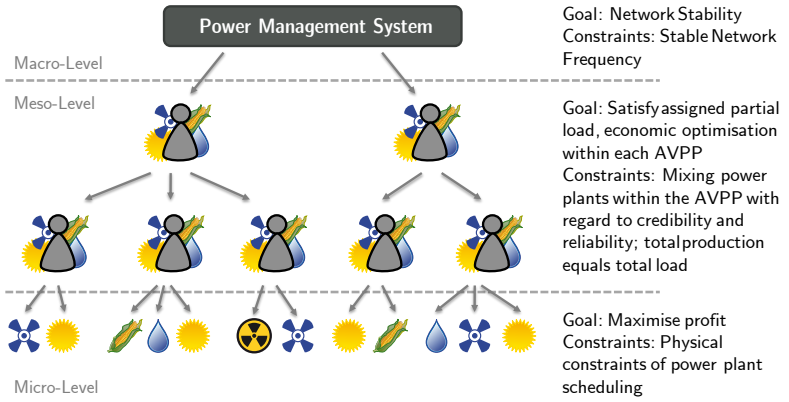


Figure 2.3: System goals and constraints on the different levels of analysis in the power management system. Different stake holders in the system (e.g., operators of power generators and distribution network operators) pursue different goals and operate under different constraints. External regulation is in place to balance these goals and constraints and achieve the overall system goal.

To create flat hierarchies, we introduced SPADA (see Section 4.1 and Anders et al., 2012b) that can be used to create partitions of the power plants controlled by one organisation. Each partition is represented by an intermediary—an AVPP—under the control of the organisation. Each AVPP in turn controls the power plants assigned to it and takes up the responsibilities of the organisation. That means that the AVPP collects data and creates the power plant schedules. This additional hierarchy level already increases scalability as organisations now control AVPPs which in turn control the power plants. As an AVPP controls multiple power plants, the organisations need to consider less entities than before. Possible alternatives to SPADA—such as coalition formation and clustering—can, e.g., be found in (Horling and Lesser, 2004).

SPADA is a decentralised algorithm that solves the set partitioning problem (SPP). It partitions a set $\mathcal{A} = \{a_1, \ldots, a_n\}$ into $k \leq n$ pairwise disjoint subsets, i.e., partitions, that exhibit application-specific properties. For power management systems, SPADA uses criteria such as credibility and reliability of the agents to form suitable partitions. The goal is to achieve a good mixture of trustworthy and untrustworthy agents in each partition so that uncertainties can be dealt with locally and each partition is able to compensate for bad predictions on its own as well as a mixture of different energy sources and generator types to enable local balancing of demand and supply.

While a flat hierarchy increases scalability, it is not sufficient for the large number of power plants installed. In Bavaria, a largely rural German federal state with some large urban centres, there are more than 440,000 DERs installed[5]. Such a number of power plants calls for a more hierarchical structure where AVPPs can in turn be controlled by AVPPs. This is possible since the scheduling task can be hierarchically decomposed by distributing the overall load on several intermediaries which in turn distribute it on the power plants controlled by them.

We have therefore extended the SPADA algorithm with a hierarchical partitioning control (HiSPADA) (see Section 4.4 and Steghöfer et al., 2013b). It consists of a control loop that is able to

[5]Source: `http://energymap.info/energieregionen/DE/105/111.html`, visited Aug. 3, 2013

introduce new hierarchy levels by introducing additional AVPPs as intermediaries. The main driver for the introduction of new AVPPs are scheduling times. Whenever an AVPP exceeds a time threshold for the sequential tasks of collecting the necessary data from power plants, calculating their schedules, and disseminating the schedules, it creates a new intermediary level for its child agents. For this purpose, HiSPADA initialises SPADA on the power plants assigned to the AVPP with an additional constraint that a minimum of two partitions has to be created. The partitions SPADA creates based on the mentioned mixture goals are represented by new AVPPs that are in turn controlled by the initiating agent. Likewise, if scheduling times become very short, an AVPP can dissolve itself, relinquishing control of its power plants to a superordinate AVPP, thus making scheduling more flexible for the superordinate AVPP as it has more power plants available. This way, the system self-organises towards a compromise between scheduling times, partition size, and flexibility.

Existing hierarchy levels can also be reorganised such that partitions conform to the originally defined partitioning criteria. Such action can become necessary in case credibility, reliability, or other attributes of the power plants change or the load curve profile the AVPPs have to fulfil changes. Predefined hierarchies are taken into consideration by HiSPADA, and it is possible to use HiSPADA to find suitable sub-hierarchies for predefined organisational entities. As each AVPP acts as a SoS, it is possible to use trust values to assess the quality of each AVPP. The evaluation in Section 4.5 shows that HiSPADA reduces the scheduling time significantly even for small system sizes.

2.6 Trust to Manage Uncertainty

To deal with the uncertainties inherent in a complex system such as the proposed power management systems, agents require the ability to measure inaccuracies in predictions as well as deviations from expected behaviour of other agents. This allows making decisions based on past experiences and forming a model of the environment's behaviour, enabling the creation of more robust schedules and dealing with unreliable agents.

Power management systems are inherently mission-critical. Their failure has massive consequences for large numbers of people, industries, and public service. It is thus of utmost importance that the power grid is stable and available at all times. Most of the engineering work on power systems therefore aims for redundancy and reliability. However, with the transfer of control from human operators to distributed agents, there is a risk that misdeterminations or unforeseen emergent behaviour can bring the system into jeopardy. This risk is mainly associated with the quality of predictions and scheduling decisions based on them. Therefore, the objective of an autonomous power management system must be to increase predictability and minimise this risk.

One of the reasons the power system is associated with so much risk is that it is an open system. Power producers and consumers—especially DERs and prosumers such as electric vehicles—are volatile, i.e., they can enter and leave the system at any point in time. Coincidentally, their benevolence can not be assumed. As agents participating in the power grid are not controlled by a single entity and can have different implementations and objectives, it is not clear whether or not they will participate in a way that is sensible for the overall system. Even if agents behave benevolently, the uncertainties imposed by the environment need to be captured. Errors caused by these "external" uncertainties are sometimes referred to as *systematic errors* (e.g., due to badly predicted weather) and *residual errors* (e.g., due to the conditions of a power plant) (Chalkiadakis et al., 2011). The acatech study refers to this as "trustworthiness" of the information (Appelrath et al., 2012, p. 86).

We propose to use *trust* as a central concept to measure this uncertainty and to improve the information's trustworthiness. Trust is based on experiences from the past. A trust model uses these experiences to create a trust value. A trust value can, among other things, give an estimation of the accuracy of predictions, i.e., their credibility, as well as of the reliability of a system participant. With respect to autonomous schedule creation, the following sources of uncertainty exist: accuracy of weather-dependent power plants' *predicted output* (credibility); accuracy of the *predicted demand* (credibility); accuracy of *providing power as scheduled* (credibility); and *availability* of controllable power plants (reliability). The main focus in scheduling is on credibility. A credibility value is derived by evaluating existing experiences capturing the difference between actual and predicted or scheduled power or demand. This value can then be used to calculate an expected deviation from a predicted or scheduled power or demand. Now, semi-autonomous scheduling as described in Section 2.4 can be further automated: instead of relying on human operators to provide expected deviations, the system can obtain those itself from trust values. By combining predictions and expected deviations, the system

can schedule its operation based on the participants' expected behaviour[6]. Humans must now only interfere with the operation of the system and guide it appropriately in exceptional situations.

Although the use of trust values significantly increases a system's ability to deal with uncertainties, a more adequate trust model is needed to precisely measure uncertainties in open systems. That is because simple trust values can only mirror, e.g., an agent's mean deviation from predictions. Since this mean behaviour likely differs from the agent's actual behaviour, a trust value is insufficient to describe the stochastic process underlying an agent's behaviour. There might be, e.g., a solar plant that either makes very accurate or inaccurate power predictions. Based on a corresponding trust value that represents the solar plant's average prediction quality, one would expect this power plant to make rather moderate power predictions, which actually is not the case. Moreover, there are situations in which it is beneficial that an AVPP can adapt itself to more than one possible development of its environment. For instance, if an AVPP knows a number of possible future developments of the power consumption, it will be able to schedule reserve power that allows to satisfy each scenario. Our idea to resolve these issues is to approximate an agent's underlying stochastic process by a number of so-called trust-based scenarios (Anders et al., 2013a). Just like a trust value, they stem from experiences and are determined at runtime. Instead of a single value, however, there are multiple scenarios, each with a probability of occurrence.

2.7 Trust-Aware Power Markets

In current power management systems, organisations can trade power in the power market with different trading modes, depending on the time horizon for which the contracts are concluded. The European Energy Exchange (EEX)[7] allows trades with a volume of at least 100kW. The trades themselves are regulated by European law and contract disputes are settled within the external legal framework.

To facilitate access to pwer markets—especially to allow AVPPs to optimise their economic efficiency on the one hand, or to balance power production and consumption on the other hand—we propose the introduction of *trust-aware power markets*. These markets allow agents to trade even small amounts of power in a secure system in which uncertainties are mediated by the trust mechanisms of the market place.

In a hierarchical SoS as described in Section 2.2, there is a need for exchanging energy within an organisation on different levels to balance power production and consumption, and thus to increase the system's robustness and efficiency. We propose that AVPPs controlled by an organisation govern an AVPP-internal power market where subordinate power plants and AVPPs can trade power intra-organisationally. An AVPP that needs additional power to satisfy future consumption could thus buy it within its organisation. As the organisation controls this internal market directly, smaller power plants can be granted access to the market, resulting in an increased number of participants and contracts.

When trading energy on power markets, agents have to cope with uncertainties similar to those which arise in the context of scheduling (see Section 2.4 and Section 2.6). These uncertainties are mainly due to information asymmetry (Ba and Pavlou, 2002), meaning that the parties have only incomplete knowledge about each other and different information about the product. In current over-the-counter markets, human operators therefore ultimately decide on the trustworthiness of trading partners when concluding contracts. Furthermore, a complex legal system ensures that the interests of all traders are safeguarded and incentivises traders to comply with their contracts. This legal system is especially important in situations in which, due to the trading procedure, market participants do not have information about or influence on which agents could or will become their contractual partners. This is the case, e.g., in uniform price auctions that are, e.g., used in the EEX's day-ahead market.

An AVPP-internal market is therefore equipped with a social system based on trust and reputation, complementing the existing legal system. Since trust and reputation stem from established contracts and their fulfilment, such a system provides mechanisms to identify untrustworthy and uncooperative agents and to effectively lower their utility by sanctioning misbehaviour and limiting their access to the market. If market participants are informed of how misbehaviour is sanctioned and the sanctions equate to a reduction in benefit, it is in the agents' best interest to behave benevolently and adhere to the rules of the internal market place. For market participants to be able to take advantage of trust

[6] This refers to forecasting systems—Technology Area 10 in (Appelrath et al., 2012). While accurate forecasts are important, there will always be uncertainty involved which becomes quantifiable with the techniques proposed here.

[7] `http://www.eex.com/`

and reputation values, the internal market implements a first-price sealed-bid auction in which bidders have information about the identity of the agent that requested the market to start the auction.

While the literature presents various approaches for preventing strategic misbehaviour and gambling through pricing mechanisms (e.g., Chalkiadakis et al., 2011; Vytelingum et al., 2010; Dash et al., 2007), agents also have to be able to identify and cope with basically benevolent agents that unintentionally show non-beneficial behaviour (see Section 2.6). Trust-based techniques allow participants to deal with both kinds of non-beneficial behaviour. These have to be integrated into the market's social system, the trading procedures, as well as the market participants' decisions.

With respect to the internal market, misbehaviour is, in principle, sanctioned directly in the short- as well as indirectly in the long-term. Direct sanctions are imposed by the organisation, for example, in the form of punitive fines. Indirect sanctions are imposed by the organisation by decreasing reputation values on the one hand, and by contractual partners by decreasing trust values on the other hand. Combined with reputation- and trust-based decisions, these measures decrease a misbehaving agent's utility. Because the risk associated with a transaction is related to the amount of traded goods, the organisation limits access to specific market products or restricts the tradeable volume within a specific time frame dependent on the agents' reputation value. Further, an organisation specifies a minimum reputation value as a prerequisite to take part in the internal market.

In a trust-aware market setting, both contractors are blamed if a contract is not fulfilled. All traders therefore try to lower their risk associated with contracts by preferably concluding contracts with trustworthy agents. Agents that ask the internal market to start a new auction specify a lower bound for a trading partner's reputation value. Bidders also specify a minimum reputation value and decide on the price they are willing to pay or demand depending on an agent's trustworthiness and reputation, resulting in price premiums and discounts for trustworthy sellers and buyers. The internal market's trading algorithm incorporates this information so that contracts are concluded in the interests of all parties. Apart from trust values, it is also possible that agents base their decisions on trust-based scenarios (see Section 2.6 and Anders et al., 2013a).

2.8 Delegation of Control with Electronic Institutions

Finally, to deal with a changing regulatory environment and to accommodate changing business goals and rules, we propose to establish electronic institutions that guide the behaviour of agents in an adaptive way. Electronic institutions can impose regulatory and organisational rules within the hierarchical structure of AVPPs and power plants.

So far, the individual power plants as well as the AVPPs are under the control of individual organisations. These legal entities govern the behaviour of the entities they control by an external legal framework. It contains rules on how the agents can participate in the power market, whether schedules should be created in risk-averse way, and guide the formation rules for AVPPs. These rules are represented in product requirements documents or other documentation and the implementation of this legal framework is the duty of the developers of the agent implementations or part of the customisation process of the power plants' control software. However, it is very hard to change those rules at runtime, e.g., when company policy changes or new features are deployed in the system.

Each organisation in the system can establish an electronic institution that embodies business rules and general behavioural guidelines. These guidelines or behavioural *norms* (Boella et al., 2009) influence the decision making process of the individual agents by providing guidance, e.g., on how an agent should behave on the power market, which optimisation goals to use for the scheduling process, or how AVPPs are formed. Electronic institutions also have to control adherence to the norms and sanction offenders accordingly with a *normative framework*. This is similar to the observation and sanctioning with with implicit norms in the market place (see Sect. 2.7). In any case, the sanctions have to be punitive, so the benefit of the agent decreases when norms are violated. Fines or lowering the reputation value can be appropriate measures. While the market—as it has been introduced earlier—uses implicit norms to moderate the participants' behaviour, using an explicit representation of norms and the sanctioning mechanism already in place institutes the market as an electronic institution as well.

Another limitation of the organisation-centric view is that all control lies with the organisation. Only the organisation has the power to give permission for certain actions or to punish misbehaviour. The key to scalable organisation structures is, however, delegation. If the organisation has to control all actions of its subordinates, it requires a global view on the system, introducing the same issues that were sought to be solved with the hierarchical structuring into AVPPs (see Section 2.5). Norms can

alleviate this problem, too, as they are also a means for delegation of control (Artikis et al., 2009). Organisations usually reserve the right to perform certain actions, e.g., to participate on the power market. Delegating norms can be used to give subordinate entities the possibility to do the same. An AVPP representing a large utility could, e.g., grant access to the power market to an AVPP representing a smaller subsidiary or even an AVPP that has been created by hierarchical self-organisation. Both aspects of normative systems for power management applications are not covered in this thesis and will be the subject of forthcoming research.

2.9 Complementary Case Study: Self-organising Emergency Response System

The autonomous power management system is an excellent case study for a system that requires very little human intervention and has a wealth of different adaptation mechanisms that work in unison to achieve a complex system goal. Large-scale open self-organising systems can, however, have properties that are not well represented in this example. To demonstrate the universality of approaches developed in this thesis, we therefore introduce a second, smaller case study that includes these additional properties and whose requirements demand a different design and a different focus.

The goal of the *Self-organising Emergency Response System* (So-ERS) is the coordination of relief units during large disasters and to act as a decision-support system for command staff. For this purpose, each firefighter, police man, medic, soldier, and officer is equipped with an Emergency Response Assistant (ERA), a hardware device connected to a number of sensors on the bodies of each member of the rescue squad[8] as depicted in Figure 2.4. The ERA provides the link between the squad leaders (e.g., company lieutenants and captains) and the respective squad members (e.g., firefighters). Information collected by each ERA is forwarded to all stakeholders that require this information, e.g., to the squad leader or other team members. The squad leader's ERA collects and aggregates this information, visualises it for the squad leader, augments it with information from the squad leader's sensors and in turn forwards it to the squad leader's superior. This way, aggregated and abstracted information goes up the chain of command. Likewise, orders are issued by command staff and distributed to subordinates. Each subordinate can detail the order and refine it to suit the situation in the field and the capabilities of its team. This way, high-level orders become detailed on their way down the chain of command. Figure 2.5 shows an exemplary system structure of such a system.

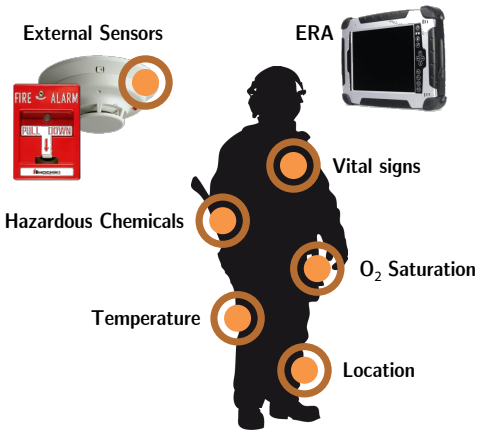


Figure 2.4: Equipment of a firefighter in the So-ERS scenario. The firefighter carries a number of sensors in its suit, including ones for his own vital signs and ones for environmental conditions. A body area network connects these sensors to the ERA which gathers and processes the data. Data from external sensors, such as fire-alarm systems can be used as well.

Characteristics and Requirements of the Self-organising Emergency Response System

The Self-organising Emergency Response System of course shares many features of the autonomous power management, such as scale and longevity. In some respects, however, it differs from the requirements detailed in Section 2.3 and it adds a number of additional ones. The following list gives an overview of the requirements and the peculiarities of the system. Especially the strong involvement

[8]For the sake of simplicity, we will in the following use the term "fire fighter" for any kind of member of a relief unit, even if it can be any other of the aforementioned specialists.

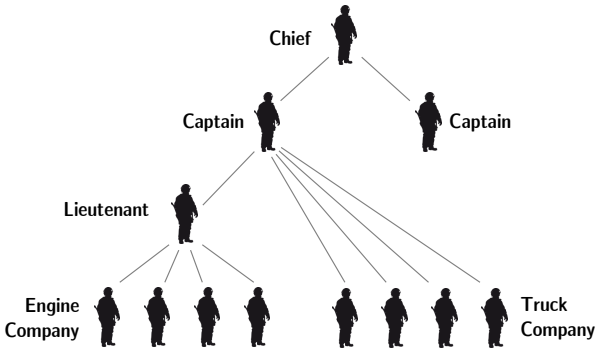


Figure 2.5: The lower levels of the chain of command in a fire brigade in the US. An engine is traditionally a water pump, whereas a truck is a versatile vehicle that usually has a ladder and various equipment. The engine has a lieutenant, drivers, and a crew of fireman that secure water supply at the scene of the incident and fight the fire directly. The truck's crew operates the specialised equipment, performs rescue missions, and support missions such as opening doors or ventilating a building. These firemen are also trained to react to dangerous chemicals or other dangers. The captain commandeers a truck and an engine company and reports to the chief.

of the users in the system and the different approach to self-organisation are clear distinctions to the previous example.

Self-organisation process is controlled by humans, not algorithms: Relief units are organised in a very strict chain of command that is determined offline by the organisational structure given within the different branches of the civilian authorities. However, this structure can be changed and adapted, e.g., to accommodate member of different branches working together. While algorithms can make suggestions for such changes based on information about the current situation and about capabilities and equipment of different units, the decision to change the structure lies with the command staff. The system thus acts as a decision support system.

Reliable communication: The function of the So-ERS depends on reliable communication links between the ERAs of the units in the field. If communication is prevented, information can not flow up the chain of command and orders can not flow down. It is thus essential, that communication is maintained even in adversary conditions. This is first and foremost a hardware requirement but can influence the design of the software as well.

Uncertainty about correctness of sensor data and conflicting information: The information collected by the individual ERAs is mostly sensor data from sensors worn on the body of the fire fighter. The sensors, however, can be faulty and due to the volatility of the environment, the sensor readings from fire fighters close to each other can differ quite a bit. The system must be able to detect unreliable sensors and conflicts in the data provided. Orders issued by command staff can also be conflicting and it must be possible for the team leaders and members of the rescue squad to deal with such situations.

Information flow and aggregation: As information is propagated up the chain of command, it must be aggregated and abstracted so that command staff is able to process the information coming in from the subordinates. In the process, no essential information must be lost and no ambiguities should be introduced. On the other hand, orders issued on higher levels will be rather abstract and will have to be refined as they are handed down the chain of command. While a high-level order could be "search sector 2A for survivors and rescue them if possible", the order on the level of the individual fire fighter will be much more specific, e.g., "check for survivors under this pile of rubble". Orders are thus refined in terms of time scale, location, and task performed.

Focus on user interface: Relief unit and command staff need fast access to the information gathered by the ERA and orders coming in from superiors. The user interface has to work well even in situations in which it is impossible to use any kind of manual input or to read a display. Command staff needs quick and intuitive access to all relevant information and have as simple way to enter orders.

Self-interested system participants: A concerted effort requires that teams and squads stick to orders that have been given by command staff with better information about the big picture. However, as the individual firefighter or rescue squad is at the scene and is confronted with a situation that changes quickly and unforeseeably, a high degree of autonomy of the individual rescue squad is required to react to the volatile environment quickly. This includes deviating from

orders, e.g., to save a life, or improvising if necessary. In this system, such behaviour is not only expected but required and has thus to be embraced.

System scale: Large natural disasters, e.g., earthquakes, require the deployment of thousands of relief units in a radius of several dozen or hundred of kilometres. The units have to achieve a coordinated behaviour, even if they maintain the high autonomy hinted at above. The information collected and generated during the disaster must be available in the aftermath to perform post-mortem analyses and to guarantee accountability for the actions during the relief effort.

Observation and control: The ERA effectively observes the actions of the firefighters and of command staff. As in the autonomous power management example, relevant data has to be observed and propagated in the hierarchy. Control decisions are made by humans in the So-ERS, but they still have to reach their recipients and the issuer must have the authority to give the order in question.

Longevity of the system: As the So-ERS will usually be deployed in a state-owned and state-operated environment, longevity and robustness to changes in the regulatory and technical environment are essential. The development of a system fulfilling the requirements outlined here would be a very pricey and time-consuming effort and can only be justified with a high degree of usefulness and a long system lifetime. Its deployment requires investments in hardware as well. During the lifetime of the system, this hardware will change. More sophisticated sensors and computing equipment will become available and new wireless communication techniques will be deployed. Such changes have to be followed up and incorporated into the system. Changes in the command structure as well as of bylaws and regulations of the authorities involved have to be dealt with, too.

Solution Approaches

In the context of this thesis, many of the requirements stated above will be ignored since they are not a consequence of large-scale open self-organising systems. The focus on the user interface, e.g., is important as soon as the user is involved but has no direct consequence on system organisation or algorithmic approaches to structuration and adaptation. For the problem of usability in adversary environments as stated above, speech synthesis for orders and important information could be a possible solution. The important aspects are, of course, system scale, the self-organisation process and the representation of its results in the system, information flow, uncertainty, observation and control, and the longevity of the system. A detailed description of the scope of the Self-organising Emergency Response System and the requirements it is designed to fulfil are provided in Chapter 14.2.

The usefulness of autonomous agents in such scenarios has been widely acknowledged (e.g., Fiedrich and Burghardt, 2007; Schoenharl et al., 2006; Rosen et al., 2002). Their application can be distinguished in two different areas: agent-based simulation (see sidebar on page 21) and agent-based decision support. The So-ERS is mainly concerned with the latter area, where agents collect, aggregate, and abstract data, present sensible decision alternatives to the users, and thus support command staff as well as first responders with all relevant information required to deal with the situation at hand. Fiedrich and Burghardt (2007) list a number of requirements agents in such a system have to fulfil, many of which corresponds to the ones outlined above. The authors also stress the ability of agents to adapt the autonomy they exhibit (i.e., how much decision making is delegated to the agent) depending on the concrete task and the possibility to integrate trust-based collaboration. Proposed MAS for emergency response include the WIPER system (Schoenharl et al., 2006) that is mainly concerned with decision support for mitigation strategies and information collection based on data gathered from cell phone towers that allows it to detect the presence of people and anomalies such as traffic jams. Based on this information it can propose strategies such as evacuation routes. The agents in WIPER act autonomously in the sense that they exchange and aggregate information without explicit human order.

Many publications stress the fact that information flow and giving users the ability to access the information they need is key to an effective emergency response system (Turoff, 2002; Jennex, 2007). An explicit study on the effects of information sharing (or the lack thereof) has found that institutional and technological obstacles influence the efficacy of an ERS (Bharosa et al., 2010). An autonomous emergency response system has to address the technological obstacles, e.g., deal with the mass of information and provide the level of detail the individual user actually requires at this point in time.

The allocation of relief units to areas can be regarded as a resource allocation problem (Fiedrich et al., 2000), similar to the creation of schedules in autonomous power management. The mitigation

strategies proposed by WIPER are also the result of a simulation and optimisation process (Schoenharl et al., 2006). These algorithms are, of course, subject to the same scalability problems as the scheduling process for power plants. Whether they can be hierarchically decomposed is, however, an open question and will not be addressed in this thesis.

Quite a few authors address the evolution of emergency response systems, a development similar to what can be seen in the power systems domain. This includes descriptions of the historical development of ERS, e.g., by Turoff (2002), and visions of ideal future systems that make the best use of current technology as well as technology yet to come. Rosen et al. (2002) present the vision of a distributed system for response to natural or artificial healthcare disasters, such as an attack with smallpox, a biological agent that has been eradicated as a naturally occurring disease but could be weaponised in the future. The system is based on decentralisation, compartmentalisation, horizontal information flow and decision support with autonomous agents. Users, especially command staff is integrated with tele-presence technology, allowing fast exchange of information and mutual decisions without the necessity of being physically at one place. The authors propose a number of technologies that need to be employed in such a case, including ad-hoc wireless sensor networks to establish reliable high-bandwidth network connections. Kyng et al. (2006) mainly address the interactivity aspect of future ERS, i.e., the way information and choices are presented to the user. The authors define a number of challenges regarding the availability and flow of information and the equipment used in emergencies. With regard to the technical infrastructure, the main challenges are which equipment and systems are available in any situation, the reliability of said equipment, and its immediate usability. The design visions developed to address these challenges include the use of wireless equipment and "dynamic support for situational overview", in effect a distributed, networked set of devices that aggregate and exchange data, as well as pre-process this data to present it to the user.

Agent-based emergency simulation
The simulation of emergency situations and disasters can be an important tool in training as well as post-mortem analysis of an actual event. It is, however, highly complex to simulate all relevant components of an emergency, including, e.g., individual buildings affected by fires, traffic flow, emergency vehicle response, and the crowds in the affected area (Jain and McLean, 2003). Agent-based models are used to simulate the different stakeholders and their behaviour, most notably, of course, the people affected by the disaster. An example for such a simulation environment is DrillSim (Massaguer et al., 2006), used, e.g., to simulate the efficacy of evacuation procedures with agents representing evacuees and emergency personnel.

Scope of the Case Study

In the context of this thesis, we will emphasise characteristics of the case study that pertain to large-scale open self-organising systems. In particular, *human-guided self-organisation* and the changes such a structuration process implies on the *information flow* will be regarded. This focus correlates strongly with *system scale* as it must be possible to create and maintain information flow structures for systems of hundreds and thousands of involved personnel. *Observation and control* in such a large system will also be in the focus of the study, especially with regard to delegation and decomposition. Finally, the design of the system will have to support its *longevity* and *openness* by providing interfaces to changing sensors, provide a scalable infrastructure for information aggregation and distribution, allow different equipment to be connected to it and embrace the inevitable changes that the system will undergo during its lifetime.

Chapter Summary and Outlook
This chapter introduced the two case studies used in the rest of the thesis to illustrate the developed techniques. The autonomous power management system will be the main example since it relies heavily on self-organisation and combines a number of adaptive mechanisms to maintain system stability and provide optimised operation. The self-organising emergency response system with its marked involvement of the human users will be revisited in Part V to illustrate the application of an agent-oriented software engineering approach to the system class.

Part II

Hierarchical System Architectures and Self-Organising Hierarchy Formation

System architecture paradigms and self-organisation algorithms for large-scale systems with hierarchical structures.

Chapter 3

Systems of Systems and Hierarchical System Architectures

Summary. In large-scale systems in which thousands of agents participate, hierarchies are a way to deal with complexity and issues of scalability. At the same time, feedback loops embedded in observation and control systems are required to detect changes in the system and to adapt to them accordingly. This chapter introduces hierarchical system architectures—with a focus on systems of systems, which provide compartmentalisation and modularisation—and the Observer/Controller architecture that operationalises feedback loops within hierarchies.

The formation of hierarchies structures information flow and control in large systems. In economics and social systems theory, hierarchies arise since it is impossible to communicate all information that is necessary to make global decisions and since it would take up too much time to control an organisation in the finest granularity from the top (Mookherjee, 2006). Instead, decision making authority is propagated down a hierarchy so that the higher the level within the hierarchy, the more abstract the decisions. This paradigm has long been a driving foundation of cultural evolution as well. As communities become more complex, the level of hierarchical organisation increases and the tasks within the community become more specialised. At the same time, a middle layer forms between the leaders and the populace that refines the directives specified from above and makes decisions according to these norms. This universally valid principle is the foundation of *Hierarchical Systems Theory* (Smith and Sage, 1973), a specialisation of the general systems theory that aims to explain the formation of hierarchies and the behaviour of the participants in a hierarchical system.

The same problems that occur in social systems can be found in computational systems as well. Since the information handled in such systems can be even bigger than the cognitive capacities of humans and the often low-tech communication infrastructure in social systems permit, the issues are even exacerbated. The divide-and-conquer approach applied in organisations can also be applied here: hierarchies are useful to mitigate complexity by dividing computationally expensive reconfiguration tasks among the agents in the hierarchy. Frameworks such as *hierarchical map-reduce* (Luo et al., 2012) therefore use the powers of hierarchical task-decomposition and existing hierarchical structures in computing infrastructure to deal with the complexity of computational tasks.

In the context of this thesis, the more interesting aspect is, however, not the use of existing hierarchies but the formation of hierarchies that serve the purposes of the system. The hierarchy that is formed should be robust w.r.t. disturbances originating within the system or the environment, adaptive w.r.t. its purpose and the role of the agents organised in it, and flexible w.r.t. the tasks that can be solved by it. As the systems in the system class we are interested in may follow pre-existing organisational structures, parts of the hierarchies can be predefined. Within this skeleton hierarchy, however, new hierarchies can be formed and adapted, yielding mixed architectures with pre-defined and self-organised portions of the hierarchy. Self-organisation is hereby defined as the formation of structure from local interactions (Heylighen et al., 2001). A consequence of this is that isolated systems with a changing organisational structure are always self-organsing (Shalizi et al., 2004). It is implicitly assumed that a state in which the system is organised is stable as well, and therefore corresponds to an equilibrium state (cf. Crefchap:stabilisation:equilibria). This only holds, however, if the process that controls the self-organisation does indeed work towards a stable system and there are enough degrees of freedom in the system that allow changing the organisation accordingly.

Before Chapter 4 details how hierarchies self-organise, this chapter introduces the basic principles, system architectures, and the embedding of feedback loops. In Section 3.1 we review different hierarchical organisation models that have been used in computational systems. Section 3.2 introduces *Systems of Systems* (SoS), a concept that goes beyond hierarchies as it considers systems that are itself composed of sub-systems where the sub-systems are opaque and have to be considered black boxes. SoS are a useful concept not only because they enforce compartmentalisation but also because they correspond to the realities of many open self-organising systems in which sub-systems of unknown and diverse providence interact with each other. Section 3.3 then shows how feedback loops can be embedded in hierarchical systems. Finally, the dependencies between the formation of hierarchical structures and observation and control on different levels of the system are discussed in Section 3.4.

3.1 Holarchies, Hierarchies, and Systems of Systems

Horling and Lesser (2004) discuss a number of organisational paradigms for multi-agent systems, including a number of flat organisational structures such as coalitions and teams. Most importantly, they identify holarchies and hierarchies as the main approaches to deal with complexity and scalability issues. For the purposes of this thesis, the ideas of holarchies and systems of systems are regarded as equivalent, but it is worthwhile to identify commonalities and differences from the liteature.

Holarchies are a special kind of hierarchical organisation. The concept of a "holon", originally described by Koestler (1967), refers to a recursive, self-similar structure, while a holarchy is a hierarchy of self-organised holons. In modern uses of this concept, e.g., for holonic manufacturing systems, a holarchy is defined as a system of such holons cooperating to achieve a common goal (Colombo et al., 2005). Holons provide a way of abstracting from granular levels of the system and thus decomposing the design effort (Cossentino et al., 2010). Depending on the level of analysis, a holon is either a non-decomposable entity or a composed agent, consisting of many underlying agents. During the development process, a holon can be refined, introducing a more intricate level of detail when necessary. When documenting and analysing the system, the same technique can be used. For the purposes of other entities in the system, each holon represents a black box, i.e., its exact internal composition is unknown. A holarchy used in such a context is usually not changed at runtime but predefined by the designer to meet the specific system requirements.

Similarly, many self-organising systems use predefined, *static hierarchies* to reflect existing hierarchical structures. In the Organic Traffic Control (Prothmann et al., 2011) project, e.g., hierarchies consist of individual traffic lights, intersections, and entire roads. On each level, the system is able to learn traffic patterns and therefore adapts to the traffic flow. The different levels allow recognition of patterns on different scales and optimise, e.g., to create green waves during rush hour. They also allow propagating information up the hierarchy and aggregate it at higher levels. This allows a more global view and thus more optimised decisions. Of course, it is often undesirable to collect a true global view due to the quantity of information and the complexity of global solution models (cf. Section 3.3). The autonomous power management system allows incorporating pre-defined hierarchies and self-organises hierarchies that optimise for a balance between computational complexity and hierarchy depth.

Hierarchies also play a crucial role in the emergence of new functionality. The artificial life community has been looking into the synthesis of dynamical hierarchies (Lenaerts et al., 2005) that show different emergent behaviour at various scales. Another research direction are hierarchies that self-assemble without an explicit algorithm (Dorin and McCormack, 2003) such as the one presented here. While these approaches are very interesting, their usefulness for large-scale technical systems has not yet been proven.

Holarchies and hierarchies are directly related to *systems of systems*. SoS are composed of systems that are themselves complex systems. They are usually distributed in nature and very large (Kotov, 1999). A lot of work on SoS originates in a military context (see, e.g., Manthorpe, 1996) where the interconnection of different, complex systems is a must to provide battlefield information and control of a wide array of weapon systems and sensors. Although these systems are heavily connected, they remain independent in many ways. Key characteristics of SoS thus include functional and administrative independence of sub-systems and geographic distribution (Sage and Cuppan, 2001). Furthermore, the behaviour of SoS is often emergent and its development evolutionary. This definition also applies to both case studies used in the context of this thesis and to many other open, heterogeneous systems. The exact definition of SoS and its application to the case studies is discussed in the next section.

power management system In discussing systems of systems and holarchies, it is often useful to distinguish the *level of analysis*—a term borrowed from sociology but widely adopted in computer science and especially in the self-organising systems community (see, e.g., Schmeck et al., 2010; Zambonelli et al., 2001). At the *micro-level* are the smallest systems that play a role in the deliberations, e.g., power plants in the case of the decentralised power management system or first responders in the case of an emergency response system. While these elements can be further broken down, their constituent elements do not have to be regarded in the system of systems since their "host" defines the external interface to them. These "hosts" thus form the lowest level of the hierarchy and are regarded as black boxes. The *meso-level* is formed by the systems that are composed of micro-level systems and other meso-level systems. In the decentralised power management example, the AVPPs form the meso-level; in the emergency response system, the command units and groups are located on this level. Finally, the *macro-level* of a hierarchy of systems is the top-level on which the overall system functionality becomes apparent. The levels of analysis are of course dependent on the point of view of the analyst. In many cases, the macro-level as perceived by the analyst will be on the meso-level when the system is really regarded as a whole. For instance, in decentralised power management, when the structure of AVPPs and power plants controlled by one organisation, such as a distribution network operator is regarded, this section of the overall system becomes the system under analysis with the rest of the AVPPs, power plants, etc. as part of the environment. While such a view is helpful to reduce complexity and not burden the analysis with unnecessary clutter, one has to be mindful when discussion systems of systems of the system boundaries and levels of analysis perceived by different stakeholders.

Both hierachies and holarchies are hierarchical structures that embody modularisation and allow to view their entities as black boxes on the meso-level (i.e., externally) and as composed of a complex structure on a micro-level (i.e., internally). When looking at the external view, we regard distinct systems, when looking at the internal view, we regard the sub-systems the system is composed of. This hierarchical decomposition is also useful since it lends itself to *hierarchical control theory*. Similar to the command hierarchy established by the relationships between superiors and subordinates in the emergency response system, a computational control system can be composed into a hierarchy with each level in the hierarchy having specialised functions, the power to make certain decisions, and the duty to carry out decisions made by an entity higher up in the hierarchy. Such control hierarchies have been used effectively, e.g., in power management systems (Schweppe and Mitter, 1972) or cognitive robotics (Gat, 1998; Mösch et al., 2006). In many cases, control decisions on higher levels also have different time horizons than ones made on lower levels. This is true, e.g., in the hierarchical control structure of the Organic Traffic Control system in which higher-level control changes traffic light patterns in the long term while lower levels make more immediate decisions. A similar decomposition can be found in classical power management (Schweppe and Mitter, 1972).

Hierarchical decomposition can also apply to the task a system has to fulfil. In hierarchical map-reduce (Luo et al., 2012), tasks are broken down into the smallest possible units that can be worked on by mapping a simple function on data. The results of this mapping are reduced—essentially aggregated. This can be done iteratively. A global reduce function can combine previously reduced results. This allows massively parallel execution of data manipulation and aggregation tasks within large server farms. The same principle of hierarchical task decomposition is used in hierarchical contract net (Kinnebrew and Biswas, 2009) in which groups of agents iteratively negotiate the handling of increasingly fine grained sub-tasks. A general formulation of the decomposition of resource allocation problems (such as the power management problem) can be found in (Van Zandt, 1995).

3.2 Systems of Systems as an Organisational Paradigm for Open Self-Organising Systems

Systems of systems (SoS) are complex systems composed of entities that can themselves be complex systems. According to Sage and Cuppan (2001), there are five typical characteristics of a SoS:

Operational independence of individual systems: each individual system is able to operate on its own and fulfil a purpose independent of the other systems in the SoS.

Managerial independence of individual systems: each individual system can be acquired, integrated and maintained by a different administrative entity, such as an organisation or a person.

Geographic distribution: the entities composing the SoS are potentially distributed over a large area and able to communicate with each other.

Emergent behaviour: the SoS's global behaviour is the result of an emergent process that is only possible due to the interactions of the composing systems and is not exhibited by any of the sub-systems alone.

Evolutionary development: the SoS changes slowly and in small increments as its environment, its objectives, or the operators' experience with it changes.

These characteristics are, e.g., fulfilled in the power management case study. Each power plant can be operated independently from the others. While there is a strong coupling between generators due to the shared power network and regulatory frameworks, the economic operation of the individual plant is—to a certain extent—independent from the operation of the others. Utilities and companies manage parts of the overall power system independently from each other. Geographical distribution is actually increasing at the moment with the wide-spread installation of distributed energy resources (DERs) such as small biofuel plants, solar plants, and wind farms. The relative stability with which load and production are balanced in the network is the emergent behaviour. No single entity of the system can provide this stability in the face of load fluctuations, weather changes, and generator outages. The system as a whole, however, provides safeguards and semi-automatic control regimes. Finally, all changes that occur in the system—i.e., changes to its structure, its objectives, its operating procedures—are relatively small with regard to the entire system but drive it towards more stable and economical operation, thus constituting an evolutionary development.

Other large-scale open self-organising systems also exhibit these characteristics, e.g., emergency response systems or battlefield information systems. In the former case, the different units sent to deal with the disaster can operate independently from each other. Managerial independence is due to the fact that firefighters, police, and emergency crews are funded by different agencies within the government. They are geographically distributed and come together to work on the same cause. Their ability to mitigate the disaster comes from cooperation and the interaction between the different crews and command staff. Finally, changes to emergency response systems, be it w.r.t. their structure, their equipment, or the procedures taken during a mission are rather slow. While changes to single units can be enacted quickly, changing the overall function of the entire system is a slow process. Interestingly, Sage and Cuppan's (2001) definition hints at some of the features of the system class:

- it's openness and heterogeneity are inherent in the managerial independence of individual systems since such a feature requires systems of different provenance and under control of different entities to work together;
- it's scale is implied by geographic distribution which only adds a meaningful distinction if there is a sufficient number of systems interacting;
- the system goal is usually only reached as the result of an emergent process that is driven by the interactions between the individual entities in the system; and
- changes to large-scale open systems are necessarily small and evolutionary. Since such systems are composed of a large number of entities and exhibit intricate and complex interaction patterns, sea changes are prohibitive since they could have unintended consequences that could hamper the system's integrity and functionality. The longevity of the system is, however, only ensured if these changes occur, since the system would otherwise become obsolete in the face of a changing operating environment.

Additionally, the definition of SoS provides another important characteristic: *compartmentalisation*. This is captured mostly in operational independence: the different sub-systems can act on their own behalf, without requiring the framework of the system in its entirety to fulfil a discernible purpose. It is therefore possible, to—at least temporarily—sever the ties between a sub-system and its super-system without impeding the sub-system's functionality. The reverse proposition does not readily hold, however, as a sub-system can still be crucial to the overall system. As an example, a complex sensor system like a radar can work without the system that calculates flight paths, but not vice-versa. Thus, a sufficient amount of redundancy is required in some SoS to maintain stable system function even under failure of singular sub-systems or a loss of connectivity. Other systems do not suffer from such drawbacks. In the power management system proposed in this thesis, both the sub-system (i.e., the power plant or AVPP) and the super-system (i.e., the AVPP or the overall system) can still function if temporary loss of communication with the rest of the system occurs.

Compartmentalisation is also evident as both case studies follow the paradigm of *uniform hierarchies* (Fox, 1979): the hierarchy consists of intermediaries (or systems of systems) that have the power to make certain control decisions based on the data they have about their local environment. Decisions

are only escalated to agents higher up in the hierarchy if that local information is insufficient to make a decision. The sub-systems under the control of a super-system have no knowledge about the overall system structure or the overall system goal. This organisation avoids information overload at higher levels of the hierarchy and leverages the advantages of *regio-central* knowledge and control. In essence, local information is centralised and local decisions are made at a central point without the need for global information or decisions. While such a meso-level decision making structure can not create globally optimal solutions on the macro-level, it is expected to perform better than purely decentralised solutions that make decisions solely based on the information available to micro-level systems.

As characteristic four of Sage and Cuppan's (2001) definition suggests, the overall functionality, as observed on the macro-level, is the result of an emergent process on the meso-level. This entails that the meso- and micro-level components of the system might be unaware of the functionality and provide the required sub-functionality with no intent of reaching an overall system goal. The task to be achieved on the macro-level might require extensive cooperation, but always has to be decomposable to a degree. In such cases where a hierarchical task decomposition is possible, a compartmentalised SoS structure allows the solution of sub-problems within the individual sub-systems that contribute to the solution of the overall problem. Such a divide-and-conquer approach becomes possible if each of the sub-systems is able to solve a fragment of the problem autonomously.

The system class regarded in this thesis adds another interesting feature. The evolutionary changes that occur in the system are no longer driven solely by the system designers or operators but can originate in the system itself. Self-organisation processes can drive the development of a system structure that is suited to both the system's environment and its objectives. Such a process can tremendously increase the longevity of a system since it will—in the ideal case—adapt to changing requirements autonomously and uphold its functionality even when adverse changes occur. On the other hand, the degrees of freedom available to a system to deal with such situations also allow for unexpected behaviour and—more importantly—unwanted behaviour of such systems. Considering the application areas of self-organising systems in which safety and robustness are an absolute must, observation and control facilities need to be in place that keep the system in check, ensure that it only adapts within reasonable boundaries and that it does not compromise itself, its environment, or its users.

3.3 Observer/Controller Architectures and their Application in Hierarchical Systems

The system of systems paradigm allows compartmentalisation and modularisation, as well as separation of concerns between distinct sub-systems. When focusing on the micro-level, i.e., the level of the individual agent, such a separation is desirable, too. The agents in a complex system work together to achieve a certain purpose, defined by the system goals and the functional requirements. Adaptation and self-organisation are secondary goals that need to work in parallel to the intended functionality. It is therefore helpful to distinguish the *functional system concern* and the *adaptive system concern*. The latter concern requires specialised observation and control facilities.

A number of system architectures have been proposed to introduce such observation and control facilities into self-organising and self-adaptive systems. They commonly provide a feedback loop that uses observations of the behaviour, state, or interaction of the system components to guide a process that influences these underlying components. The components and the feedback loops form a hierarchy where the components form a system that is observed and controlled by the feedback loop. It is possible to stack these systems and feedback loops to create a hierarchy. Systems on lower hierarchy levels are usually considered black boxes where little assumptions can be made about the inner workings (see Chapter 6 for a more detailed discussion of black-box and white-box monitoring).

A prominent example of such a feedback loop and an accompanying architecture that embeds the feedback loop in autonomous systems is the MAPE-cycle and the Autonomic Computing Architecture, put forward by the IBM Corporation (2006). The MAPE-cycle, depicted in Figure 3.1, uses sensors to *m*onitor data from underlying systems—i.e., the functional system that it observes and controls—that it then *a*nalyses. If the analysis reveals a need to change the system, a new *p*lan is created that is then *e*xecuted by changing the system with the effectors available to the feedback loop. A MAPE-cycle is encapsulated by an Autonomic Manager (AM) that controls a Touchpoint—an encapsulation of a "managed resource", abstracting its actual interfaces and configuration possibilities and providing sensors and effectors—or another Autonomic Manager. The basic architecture suggests that Autonomic Managers that control Touchpoints are in turn controlled by other Autonomic Managers that provide

high-level functionality and aggregate information from several AMs. This forms a hierarchical structure at the top of which a human operator controls the system by providing procedural guidelines and policies that are effectuated by the AMs.

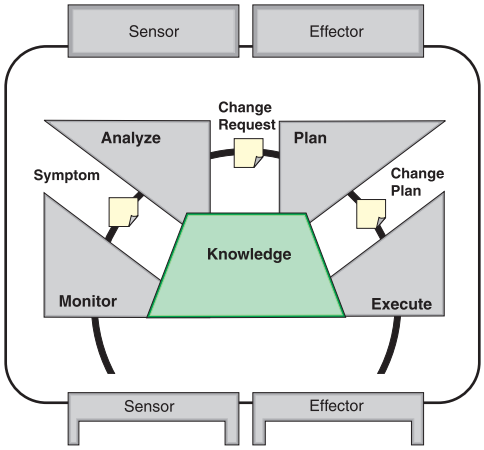


Figure 3.1: The MAPE-cycle, embedded in an Autonomic Manager (IBM Corporation, 2006). The AM utilises sensors and effectors with which a managed resource can be monitored and controlled. The MAPE-cycle provides the feedback loop that identifies undesired symptoms in the system and issues actions to change them. As an AM provides sensors and effectors as well, it can itself be a managed resource observed and controlled by an AM on a higher level.

Centralised and Distributed Observer/Controllers

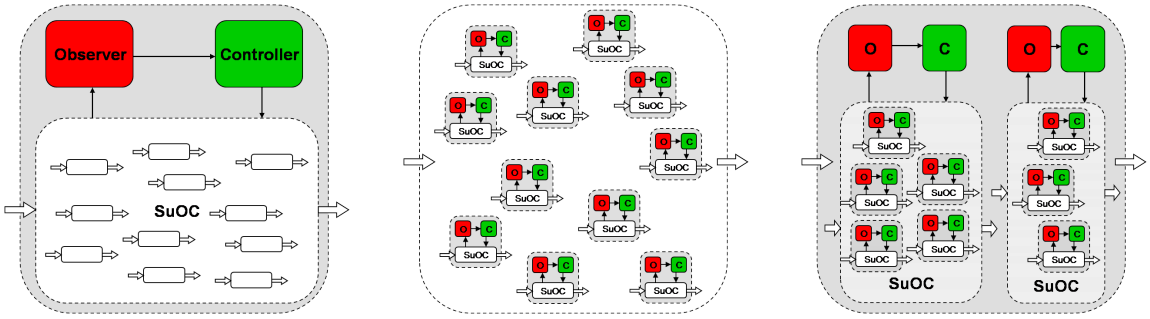
As the Autonomic Computing Architecture is focused on a specific deployment area, more generic architectural patterns have been proposed. The *Observer/Controller* (O/C) *architecture* encapsulates many of the features of the MAPE-cycle but does not prescribe the phases of the feedback loop. It also adds flexibility with regard to where the feedback loops are situated in the system. An O/C provides a feedback loop for a *System under Observation and Control* (SuOC) (Richter et al., 2006), corresponding to the functional system concern. The system itself can be composed of different sub-systems that are, in turn, individually controlled.

Different variants of the O/C are depicted in Figure 3.2. The observer part of a centralised O/C gathers data from a SuOC that can be arbitrarily complex internally. In such cases, the system is usually regarded as a blackbox, so that the individual elements of the system are not distinguished. The control actions then influence all elements alike, as they too are directed at the entire SuOC. Such a structure is useful if weak self-organisation (Di Marzo Serugendo et al., 2005) is called for in a system, i.e., the self-organisation process is controlled by a centralised entity within the system. In contrast, a distributed observer/controller architecture proposes a separate O/C for each system element. These elements, usually implemented as autonomous agents, are observed and controlled individually. In many cases, to achieve a self-organising behaviour, the O/Cs interact with each other and follow an algorithm to adapt the system to new circumstances. Examples of such distributed self-organisations include coalition formation (Rahwan et al., 2009) and set partitioning (Anders et al., 2012b). Such algorithms work with local knowledge only, i.e., they make decisions based on the state of the agent and its environments either as perceived through the cognitive faculties of the agent or O/C or by exchange of knowledge with others. As the self-organisation process is not dependent on a single, central entity, such systems exhibit strong self-organisation (Di Marzo Serugendo et al., 2005).

The systems we regard employ a slightly different version of the hierarchical Observer/Controller model depicted in Figure 3.2c. While the figure implies that an O/C directly works on a set of agents, we instead represent this set of agents with another agent—called an *intermediary*—similar to holarchies and systems of systems. The outer SuOC is thus a full-fledged autonomous agent, capable of interactions, acting in the environment itself, and making own decisions.

Regional Control in Hierarchical Systems

However, the use of local information can lead to sub-optimal decisions. Only with a global picture of the current situation can a truly optimal decision be made. As the communication overhead of information collection and the decision time increases with the number of entities in a system, scalability issues prevent most adaptive systems from using a strictly weak self-organisation approach. To ensure reasonably good decisions, a hierarchical approach as depicted in Figure 3.2c can be taken. Individual entities are still controlled by separate O/Cs, but decisions that require more information and influence



(a) Centralised Observer/Controller. (b) Distributed Observer/Controller. (c) Hierarchical Observer/Controller.

Figure 3.2: Variants of the Observer/Controller architecture. Depending on the complexity of the system under observation and control, the observer/controllers can be located on different levels. In the simplest case, the O/C observes and controls the entire systems. It is, however, also possible to equip each individual sub-system with an O/C and—as is most common—to combine both approaches in a hierarchical setup. Pictures based on (Schmeck et al., 2010).

a group of agents are made *regionally* by a higher-level O/C which collects information about the group and can exchange information with other O/Cs at the same level. Such a system structure lends itself to the definition of System of System as each SuOC is its own complex system that works as a subsidiary of a higher-level system but can function independently. Solutions of this kind are "best effort", in the sense that they make a trade-off by using all information available in a certain region of the system to find a good solution without imposing the communicational and computational overhead of compiling global models from the distributed agents.

Regionalised information and decisions, however, make the placement of the observer and of the controller crucial since it defines information flow, what can be observed in terms of agent states, behaviour, and interactions, and which agents can be controlled. The O/C has to be able to access the relevant information, monitor the interactions between the agents, derive a system state, check if the system works correctly and at the same time be able to adjust system parameters, change the system structure, or react in other ways if necessary. Luckily, principles of systems of systems can give important hints on the correct placement: sub-systems that can work independently and are highly integrated can be controlled as a unit. To the outside, they appear as a black box if the O/C is internal. Internally, they can again be composed of independent sub-systems, each controlled by their own O/C. Therefore, there is a congruence between the hierarchical composition of black boxes such as in systems of systems and the hierarchical O/C structure discussed and depicted above. While the functional parts of the system are connected hierarchically to exchange work products or data, the O/Cs that observe and control these systems can be connected as well and exchange information and command decisions. In fact, agents that take a special role in the hierarchy or represent a group of agents are natural candidates to take on the responsibilities of an O/C for those agents they control. The Autonomous Virtual Power Plants introduced in Chapter 2 are an excellent example for such agents.

Separation of Concerns between Observer and Controller

The separation of concerns between the O/C and the SuOC is an important topic in another regard: the Observer/Controller should not provide the basic functionality of the system but only controls adaptation and self-organisation. This requires a clear distinction between the system's basic functionality and its adaptation. This distinction can be made during the requirements analysis process (cf. Chapter 7) and during the formal specification (cf. Chapter 5). If such a separation of concerns is strictly enforced during the entire design process, it is even possible to roll out an application without adaptive capacities and add this functionality later on (Seebach et al., 2010). In such a case, the interfaces between the productive system and the adaptive system have to be well-defined. This requirement mainly corresponds to the parameters the adaptive control loop can and may change at runtime. Adaptations may not interfere with the productive work and the state changes during the productive phases may not void the ongoing reconfiguration. A clear separation can be achieved by putting the productive system into a *quiescent* state during reconfiguration in which it is limited to actions that will not invalidate the reconfiguration process (cf. Chapter 5).

In summary, the combination of self-organisation and architectural principles that support this kind of adaptation with the principles of systems of systems and with a hierarchical structure allow the creation of scalable, flexibly adaptive systems. Transitioning existing systems to hierarchical system architectures or to system of systems architectures requires the decentralisation of control decisions, regionalisation of information, and introduction of clear system boundaries. When designing hierarchical systems, these aspects can be kept in mind from the start and appropriate abstractions, such as AVPPs can be introduced.

3.4 Dependencies between Hierarchy Formation and Observation and Control

If hierarchies are present in a system, its scalability and adaptivity possibilities can be greatly extended, especially if hierarchies are not completely pre-defined but are allowed to evolve at runtime based on the current system state and environment. The techniques proposed in this thesis allow the dynamic change of the hierarchy to allow systems to grow and adapt their hierarchical structure to the needs of the application.

To achieve this, the different control mechanisms have to be related to each other. In case of the autonomous power management case study, an adaptation of the hierarchy occurs when certain constraints are violated. A new hierarchy level is introduced, e.g., when the time required for creating a schedule for the power plants controlled by an AVPP exceeds a certain threshold. This implies that scheduling times are observed and a constraint over them is formulated. A new hierarchy level adds structure by introducing a stronger compartmentalisation. The power plants that were formerly controlled by one AVPP are now controlled by two or more. This in turn influences the individual AVPP's ability to create schedules, e.g., because fewer controllable power plants are available. It also changes which agents are observed and controlled within the different involved AVPPs.

As soon as different feedback loops influence each other, maintaining system stability becomes much more complex. In particular, interferences between different adaptation mechanisms have to be regarded (cf. Section 8.5). If structural adaptations occur, the observation and control infrastructure has to adapt as well. These topics will be discussed in more detail in the appropriate chapters.

Chapter Summary and Outlook

This chapter introduced hierarchical system structures and systems of systems as an organisational paradigm for large-scale self-organising systems due to the advantages they provide with regard to compartmentalisation and regio-central control. The Observer/Controller architecture—especially in their distributed or hierarchical variant—supports observation and control tasks in such systems. It will therefore be used in Part III and Part IV as the foundation for the observation of system behaviour as well as control and stabilisation. To establish a hierarchy in a large-scale system, a suitable self-organisation algorithm is necessary. The next chapters introduces HiSPADA that is able to establish stable hierarchies adapted to the system's purposes.

Chapter 4

Self-Organising Hierarchy Formation with HiSPADA

Summary. Self-Organisation algorithms that establish a hierarchy based on performance considerations increase scalability based on the principle of compartmentalisation. This chapter introduces HiSPADA, a hierarchical partitioning control to create hierarchies driven by the performance of the problem solution algorithm employed in the partitions. The hierarchy formation process can use arbitrary algorithms that form a flat hierarchy by performing a partitioning on a set of agents. As one example of such algorithms, SPADA, the Set Partitioning Algorithm for Distributed Agents is outlined. The evaluation results show that the runtime is significantly reduced as compared to a centralised solution with only minimal impact on solution quality.

Publication. The results detailed in this chapter have been published in (Steghöfer et al., 2013b) and (Anders et al., 2012b).

In order to achieve the advantages of a compartmentalised system of systems as outlined in Chapter 3, the system should self-organise a suitable structure in response to changing environmental conditions and the internal system state. This process of self-organisation should form a hierarchy in which the overall system goal on the macro-level can be fulfilled while, at the same time, allowing the individual compartments on the meso-level to achieve their sub-goals and giving the agents on the micro-level the opportunity to contribute to the system and maximise their own benefit.

To ensure that the hierarchical structure fulfils these requirements, it is necessary to detect changes in the system that warrant a re-organisation. Techniques for this purpose are introduced in Chapter 6. The criteria used in the formation of the hierarchy must, of course, support the requirements outlined above. The self-organisation process must terminate and find a suitable system structure quickly. Otherwise, the system will not be able to fulfil its goal sufficiently. Thrashing, i.e., the constant creation and dissolution of compartments, and other negative effects have to be avoided as well. This has to be considered in the design of the self-organisation algorithm but also in the definition of the criteria that lead to re-organisations and that determine termination of the algorithm.

This chapter introduces two self-organisation paradigms. The first one, creates a partitioning, i.e., a division of the elements of a set into disjoint subsets based on application-specific criteria. Two algorithmic solutions for this paradigm are introduced: the **S**et **P**artitioning **A**lgorithm for **D**istributed **A**gents (SPADA), explained in Section 4.1 and the **P**article **S**warm **O**ptimizing **S**et **P**artitioner (PSOSP), detailed in Section 4.2. Their application to the power management case study and their solution quality are outlined in Section 4.3. The second paradigm, hierarchical self-organisation with the corresponding algorithm HiSPADA is then explained in Section 4.4 and a detailed performance evaluation is given in Section 4.5. Section 4.6 compares the proposed approaches with similar algorithms from the literature.

4.1 Introducing Flat Hierarchies: SPADA

SPADA (Anders et al., 2012b), the **S**et **P**artitioning **A**lgorithm for **D**istributed **A**gents, solves the set partitioning problem (SPP) in a general, decentralised manner. In the SPP, the goal is to partition a set

$\mathcal{A} = \{a_1, \ldots, a_n\}$ into $k \leq n$ pairwise disjoint subsets, i.e., partitions, that exhibit application-specific properties. For example, if the objective is to group similar or dissimilar elements together, the SPP is equivalent to clustering or anticlustering (Valev, 1998). This can be achieved by complementing the SPP with an appropriate metric. In case such a metric defines how well agents can work together on a common task, the SPP is equivalent to coalition structure generation (Rahwan et al., 2009). Due to this flexibility, algorithms for the solution of the SPP in multi-agent systems (MAS) have a broad area of application. Since SPADA has been designed to solve the SPP in general, it can be applied to these specific problems as well.

Some of the algorithms proposed to address problems like coalition formation, e.g., those formulated and solved as a linear programming problem, require a global metric or some form of global knowledge needs to be gathered before the algorithm can be run (e.g., Rahwan et al., 2009). Thus, they are not applicable if the cost of gathering this knowledge is high or the system does not support obtaining such knowledge due to its openness and adaptivity. Many existing approaches try to circumvent this problem one way or the other, but they still suffer from some drawbacks that limit their usefulness in highly adaptive systems. A crucial step in some algorithms (e.g., those proposed in Shehory and Kraus, 1998) is to distribute this global knowledge among the agents. All possible coalitions and their utility are pre-calculated and the best one is picked after a global announcement. This ensures optimal solutions but introduces global synchronization points. This is not practical in many cases, especially when agents enter and leave the system arbitrarily.

SPADA solves SPPs in a completely decentralised fashion, relying only on local knowledge. Thus, no central metric is necessary and the quality of the partitioning is evaluated locally. With different metrics, SPADA can easily be applied to different clustering and coalition structure generation problems in MAS. In the following, we give a summary of SPADA's basic functionality and characteristics. A more detailed description can be found in (Anders et al., 2012b), along with a comparison with the algorithm SPADA has been inspired by (Ogston et al., 2003). We use the terms "partitioning" or "reorganisation" to denote the process performed by SPADA.

Overview of the Behaviour of SPADA

All operations SPADA performs to come to a solution can be mapped onto graph operations that operate on an overlay network, which is called *acquaintances graph*. The acquaintances graph is defined by the agents participating in the SPP and acquaintances relationships between them, symbolised by directed links. To simplify graph operations that modify partitions, it is stipulated that each partition is a directed tree of marked links, which results in a directed forest for the partitioning as a whole. An example graph is depicted in Figure 4.1.

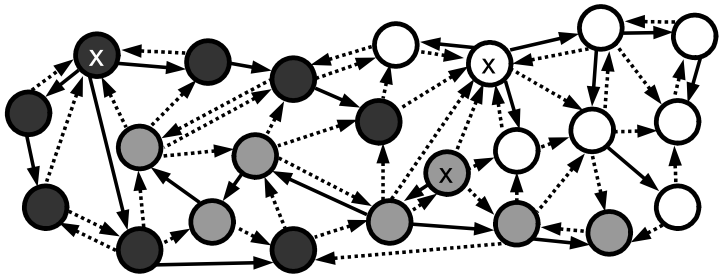


Figure 4.1: An acquaintance graph as used by SPADA and three colour-coded partitions. Unmarked links are denoted by dashed lines, marked links are denoted by solid lines. The partition leaders are identified with "x". Please note that an actual graph would be more strongly connected than the one depicted here.

A partitioning for such a MAS is a division of the acquaintances graph into several subgraphs with a pairwise disjoint set of nodes (i.e., agents). Partitions are represented by *marked links*, which are links with a partition-specific flag. Links without this flag are called *unmarked links*. A marked link (a, b) between two agents a, b states that a is acquainted with b, and that a, b are members of the same partition. Note that two agents c, d can be in the same partition without a link between them. As links are unique, a link is either marked or unmarked. In terms of the acquaintances graph, a partition is defined by a tree T of marked links with a designated agent (the *leader* λ) at its root. Each partition thus has always exactly one leader. This results in a directed forest for all partitions. With x an arbitrary agent and $\mathcal{R}^*$ the reflexive transitive closure of the relation $\mathcal{R}$ induced by marked links, $\{x \mid \lambda \, \mathcal{R}^* \, x\}$ describes the set of members of λ's partition.

The partition leader is responsible for optimising its partition according to application-specific criteria. The leader uses the control loop depicted in a simplified form in Figure 4.2. Each leader peri-

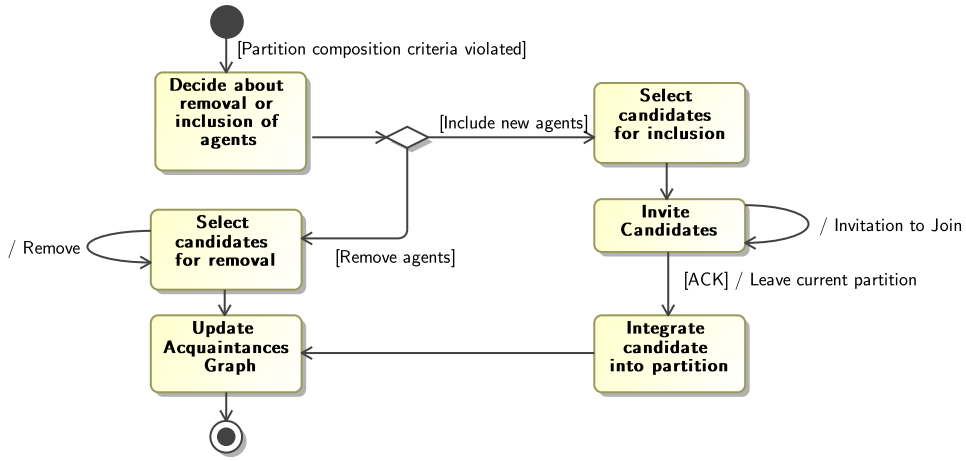


Figure 4.2: The partition leader's simplified control loop to alter the composition of a partition. If the partition composition criteria are violated, the partition leader decides whether the criteria can best be restored by including additional agents or removing agents from the partition. Depending on the decision, candidates for inclusion or removal are selected. The best candidates are informed about the decision. After the candidates have been included or removed, the acquaintance graph is updated.

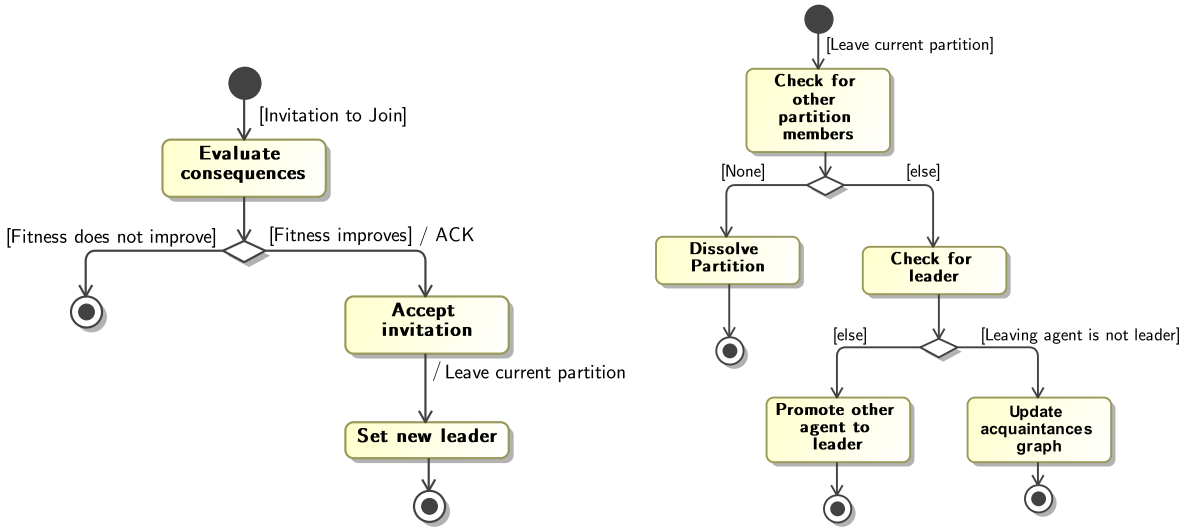
odically evaluates if it is beneficial to integrate new agents into its partition or to exclude members from it based on the local knowledge available to it. The latter can be beneficial if the partition's or an agent's properties have changed so that the partition's formation criteria no longer favour including the agent. Integrating and excluding agents is performed by modifying the acquaintances graph. Potential candidates for integration are acquainted agents that are not yet in the partition.

Initially, every agent is the leader of a partition with size one. Though every agent is basically capable of being a leader—except for this property, agents may be heterogeneous—there is only one leader per partition to avoid inconsistencies in the course of the formation process. A leader therefore unambiguously identifies its partition. It is responsible for managing the partition's composition and knowledge by modifying the marked and unmarked links of its members. For this purpose, a leader knows all marked links of its partition (the members) and all unmarked links of all members (further acquaintances of the members). A leader λ changes its partition Λ by requesting Λ's acquaintances (i.e., the acquaintances of Λ's members) to join or by asking agents within Λ to leave in order to increase the benefit of the individual partitions or the system. Because of the constant number of links per agent, the number of external agents a leader can contact to extend its partition is limited by the number of partition members multiplied by the number of acquaintances per agent, minus the number of partition members.

To decide termination, leaders periodically evaluate application-specific termination criteria formulated as predicates based on local knowledge. These predicates are constraints that can also be monitored at runtime and be used to trigger reorganisation (cf. Section 5.1 and Chapter 6). In the case study, comparisons with neighbouring partitions based on their composition are used as outlined further below. If the constraints are met and the compositions of the neighbours are similar enough, the leader marks its partition as terminated. However, termination is only decided locally. It is thus not guaranteed that the algorithm terminates globally. Intuitively, changes in partitions that are not terminated can trigger changes in terminated partitions since the non-terminated ones can a) invite members of terminated partitions to join them, changing the mixture of the terminated partition or b) exclude members, causing the neighbourhood with which the mixture is compared to change. Such behaviour can be reigned in by a corresponding parametrisation of the algorithm as discussed in Section 4.3 or by limiting the number of rounds as in Section 4.5.

As long as a partition is terminated, its structure is not changed until a member is integrated into another partition. SPADA can thus make selective changes to an existing partitioning, which is very useful in dynamic environments. The evaluations in Section 4.3 and in (Anders et al., 2012b) show that SPADA's local decisions lead to a partitioning whose quality is within 10% of the solutions found by a centralised metaheuristic.

Agents that are not leaders have to react on invitations to join a partition and on requests for a promotion to partition leader. The former case is depicted in Figure 4.3a. An agent that receives an



(a) Control flow of an agent that was invited to join a new partition.

(b) Control flow of a partition leader handling an agent leaving the partition.

Figure 4.3: The agent's decision to accept or decline an invitation to join a new cluster and the partition leader's reaction to an agent leaving a partition. Please note that both behaviours can be accomplished autonomously based on local knowledge.

invitation can evaluate its own fitness criteria based on the local knowledge it has about the partition it is currently a part of. If the fitness of both the new partition (as transmitted by the partition leader) and of the old one in sum improves, the agent accepts the invitation. It sends an acknowledgement to the inviting partition leader and informs the leader of its current partition about its intentions.

The reaction of the agent's former partition leader is depicted in Figure 4.3b. The leader first checks whether there are other members in the partition (the leaving agent can itself be the leader). If not, the partition is dissolved. If the partition has other members, the leader checks whether it leaves the partition itself. In that case, another agent has to be promoted to partition leader. Otherwise, the leader updates the acquaintance graph by removing the marked link to the leaving agent.

In many cases, it is useful that the partitions created are represented by an intermediary. In the case study, this intermediary is the AVPP. The leader of a partition instantiates a new AVPP agent after partitioning has finished. The AVPP then assumes control of all power plants in the partition.

Applying SPADA to the Decentralised Power Management Case Study

When applying SPADA to the case study introduced in Chapter 2, the partitioning criteria are selected such that an anti-clustering is achieved. Each partition represents an AVPP. An AVPP should internally have power plants from different energy sources and with different credibility w.r.t. to the predictions made. Such a mixture allows the AVPP to internally balance fluctuating production due to environmental factors such as the weather and due to uncertainty introduced by imprecise predictions. The partition's composition criteria are based on these mixture constraints[1]. A partition leader compares the mixture of its own partition with the one of neighbouring partitions. Partition size also plays a role. If the neighbouring partitions have a significantly better mixture or are significantly larger or smaller, the composition criteria are violated and the partition leader tries to change the composition of its partition as depicted in Figure 4.2.

It is important to note that these criteria are specifically designed to support the creation of power plant schedules (cf. Chapter 9). The requirements of another algorithm thus drive the definition of the constraints that determine whether a partitioning is good. The formation of structure in the system is thus used to aid the main function of the system and promote system stability and robustness.

[1]The criteria are defined as constraints on the state model of the partition leader (cf. Chapter 6).

4.2 An Alternative to SPADA: PSOSP

While SPADA is a heuristic working with local knowledge, using a central approach that also solves the SPP in general but by making use of global knowledge can potentially achieve better results. To compare these approaches, we use a particle swarm optimiser (PSO) (Kennedy and Eberhart, 1995), a meta-heuristic for finding solutions in optimisation problems, based on the flocking behaviour of birds. In principle, a PSO consists of a swarm of particles positioned in an n-dimensional search space that consists of all possible solutions. These particles follow rules for their motion through that search space that are based on a personal best position $\vec{x}_{\text{pBest}}(t)$ (cognitive component), a global best position $\vec{x}_{\text{gBest}}(t)$ (social component), random values r_1, r_2 and a constant w (exploratory component), and factors c_1, c_2 that are learned in the course of the search (self-reflection component). As each position $\vec{x}_i(t)$ a particle finds itself in corresponds to a valid solution of the problem, the particle can evaluate that position with a fitness function (Reyes-Sierra and Coello, 2006). It can also be used to calculate the velocity vector $\vec{v}_i(t+1)$ that determines the particles movement in the next time step.

$$\begin{aligned}\vec{v}_i(t+1) = w \cdot \vec{v}_i(t) &+ c_1 \cdot r_1 \cdot (\vec{x}_{\text{pBest}}(t) - \vec{x}_i(t)) \\ &+ c_2 \cdot r_2 \cdot (\vec{x}_{\text{gBest}}(t) - \vec{x}_i(t))\end{aligned}$$

PSOs can be executed synchronously with each particle updating its position once within a time step or asynchronously where each particle is a separate thread or process that updates its position as often as possible. The former variant is simpler to implement since it is easy to maintain a consistent view of the system and update $\vec{x}_{\text{gBest}}(t)$ while the latter variant is usually faster and can be distributed.

The standard formulation of particle swarm optimisation assumes a continuous search space, i.e., a search space that consists only of variables with continuous domains. As the set partitioning problem is formulated on sets of agents, however, a discrete formulation is necessary. The one suggested by Kennedy and Eberhart (1997) is applicable to discrete binary problems. García and Pérez (2008) introduced a "Jumping Frog Optimisation" (JFO) in which the particles do not follow a velocity vector but perform random jumps within the search space, jumps to approach their personal best position, or jumps to approach the global best position.

$$\begin{aligned}x_i(t+1) =& c_0 \cdot x_i(t) \oplus c_1 \cdot \vec{x}_{\text{pBest}}(t) \oplus c_2 \cdot \vec{x}_{\text{gBest}}(t) \\ &\text{with } c_0, c_1, c_2 \in [0;1] \text{ and } \sum_{k=0}^{2} c_k = 1\end{aligned}$$

The factors $c_0, \ldots, c_2$ can be interpreted as probabilities with which a movement towards one of the three positions is performed. The term $c_0 \cdot x_i(t)$ is interpreted as a random move. As a speciality, JFO allows multiple moves of a particle per iteration. Furthermore, the original formulation includes a local hill-climbing after each move to improve the position of the particle in the local neighbourhood.

Basically, the particle swarm optimiser to solve the set partitioning problem—in the following abbreviated as PSOSP—uses a swarm of particles that roams the discrete search space by modifying candidate solutions by basic set operations ("split"—that divides a set p_i with $|p_i| \geq 2$ into two non-empty disjoint set p_j and p_k such that $p_i = p_j \cup p_k$—and "join"—that merges two non-empty sets p_j and p_k into one set p_i such that $p_i = p_j \cup p_k$.). Particles contain a set P_i of sets whose elements are the agents that have to be partitioned. Each set represents a single partition. The union of the sets is the set of all agents. A move thus corresponds to the application of a sequence of split and join operations on the set of sets.

To perform a random move, the particle randomly selects whether to use a join or a split operation and then applies them on one respectively two random sets. In case a split operation is applied, the selected set must have at least two members. When a move to approach the personal or the global best position is performed, the particle determines whether a join or a split operation should be performed and which set(s) should be affected. For this purpose, the structural differences between the best positions and the current partitioning represented by the particle are evaluated.

PSOs are heuristic algorithms, meaning that they are not guaranteed to find the globally optimal solution. However, they are *anytime algorithms* that provide a valid solution—not necessarily optimal—at any point during their runtime. The PSOSP terminates based on custom termination criteria, when a certain time limit is reached, or when it is aborted by the controlling program. In contrast to other existing mechanisms like k-means clustering or the k-nearest neighbour algorithm, SPADA and the PSO do not specify a concrete number of partitions or clusters in advance.

4.3 Flat Hierarchies of Power Plants in the Decentralised Power Management Case Study

To evaluate SPADA's performance, we recapitulate and extend the evaluation from (Anders et al., 2012b). It regards the problem of partitioning the set of all power plants into *Autonomous Virtual Power Plants (AVPPs)*—i.e., groups of power plants as introduced in Chapter 2—and compares the results of the algorithm with solutions calculated by PSOSP. The aim is to achieve a similar mix of trustworthy and untrustworthy power plants in each AVPP by performing an anticlustering. The trivial partition that consists of one big AVPP, i.e., the *Grand Coalition*, is not desired.

To assess the fitness of a partitioning, for each AVPP, we calculate the mean credibility of power plants contained in this AVPP. Since the goal is to equalise these mean credibility values, their standard deviation σ is to be minimized. Because our credibility values are from the interval $[0, 1]$, σ is always between 0 and $\sqrt{0.5}$. We therefore define a partitioning's fitness $\mathcal{F}(\sigma)$ as a function of σ as follows:

$$\mathcal{F}(\sigma) = \left(\frac{1}{\sigma + 1} - a\right) \cdot \frac{1}{1 - a}, \text{ with } a = \frac{1}{\sqrt{0.5} + 1}$$

$\mathcal{F}(\sigma)$ monotonically decreases on the interval $\left[0, \sqrt{0.5}\right]$ so that we have $\mathcal{F}(0) = 1$ for optimal partitionings (each partition has the same mean credibility value) and $\mathcal{F}\left(\sqrt{0.5}\right) = 0$ for a maximum standard deviation. Expecting SPADA to perform well, $\mathcal{F}(\sigma)$ is particularly sensitive to changes when σ is small. The evaluation of PSOSP's solutions is performed with the same fitness function.

The tests were performed in a system consisting of 435 agents, implemented in Repast Simphony[2], which uses a sequential, round-based execution model. Since we avoided the complexity of an asynchronous execution model, SPADA could not benefit from parallelism. In each round, every leader could modify the composition of its partition, mix acquaintances, and evaluate termination criteria, which could result in multiple changes to the partitioning. We assume that all messages are processed correctly and all agents work properly. SPADA's local fitness function compares the mean credibility value of a partition to the mean credibility value of the partition's local environment. If the similarities are within a small threshold, a leader marks its partition as terminated. The higher the deviation between the mean credibility value of the regarded partition and the local environment, the lower the partition's fitness. As mentioned in Section 4.1, a partition's local environment consists of the partition's acquaintances that are not members of the partition. An agent's decisions thus aimed at minimizing this deviation, thereby improving $\mathcal{F}(\sigma)$.

To avoid fluctuation of agents between partitions, the reward function of a requested agent additionally implemented a mechanism that allows the agent to prefer partitions in which it has been a long time if the reward is small. Note that it an agent's local assessment of a partition's fitness and the global fitness of the partitioning appraising the result of solving the SPP can differ due to the different information taken into account. The PSO used five particles to find a nearly optimal partitioning within 30 seconds. We recorded 300 simulation runs for each test (see Table 4.1 for parameters). For each run, we generated the initial acquaintances graph at random. Further, each agent had a fixed uniformly distributed random credibility value $\in [0, 1]$. By parametrization, we prevented both algorithms from forming the trivial optimal partitioning consisting of one big AVPP ($\sigma = 0$). The following results are arithmetic means of recorded data (similar results are obtained with a beta distribution with $\alpha = \beta = 0.1$).

In the tests $t_1, ..., t_5$ (see Table 4.1 for parameters), both algorithms modified an initial partitioning in which each agent formed its own partition. In the first simulation runs for t_1, ..., t_5, both algorithms modified an initial partitioning in which each agent formed its own partition to identify suitable parameters for SPADA. The results are depicted in Figure 4.4. PSOSP achieves a nearly optimal mean fitness of 0.999, and SPADA also performs very well: the mean fitness increases rapidly to a value beyond 0.9 (however, please note that the partitioning is changed several hundred times in each round), and slowly converges to mean values between 0.856 and 0.959.

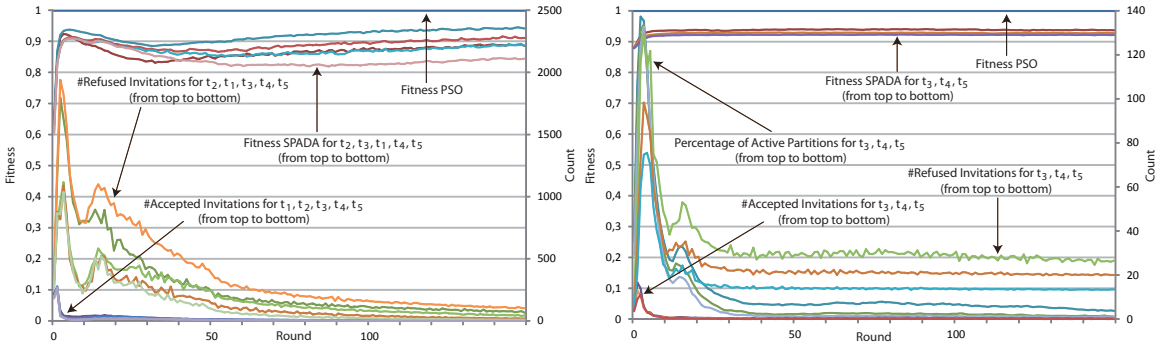
For each test, the graphs in Figure 4.4a show similar characteristics: the number of accepted and refused invitations drops rapidly because partitions terminate over time[3]. Some partitions are reactivated by active partitions, leading to a temporal decline in fitness. However, since memberships are rearranged, SPADA achieves a higher fitness value for suitable parameter settings afterwards (t_2 and

[2] `http://repast.sourceforge.net`

[3] Regarding a partition Λ, the number of messages necessary to identify suitable candidates for integration is $O(n \cdot |\Lambda| + m_r)$, for exclusion $O(n \cdot |\Lambda| + m_e)$, and to handle join partition requests $O(n \cdot |\Lambda|)$.

Test	#Links per Agent	#Links to Rate	τ^r
t_1	10	20	-2
t_2	20	20	-2
t_3	20	10	-2
t_4	20	10	0
t_5	20	10	1

Table 4.1: Different sets of parameters used for evaluation. We varied the number of links per agent, the number of links rated by leaders to identify candidates for integration, and the minimum reward τ^r necessary to become a candidate. Tests t_1 and t_2 were exclusively used to assess performance when forming an initial partitioning, whereas tests t_3, t_4, and t_5 were also used to evaluated SPADA's reconfiguration behaviour.



(a) Evaluation results for an initial configuration.

(b) Evaluation results for reconfiguration.

Figure 4.4: For evaluation, we varied the number of links n per agent, the number of links m_r rated by leaders to identify candidates for integration, and the minimum reward τ_r necessary to become a candidate as detailed in Table 4.1. The graph shows SPADA's average performance for tests t_1, t_2, t_3, t_4, and t_5 compared to the PSO when modifying an initial partitioning and when reconfiguring an existing one.

t_3). Looking at the results in detail, for t_1, the fitness is rather low and the number of refused invitations high, which is not desired as this means unnecessary processing and message delivery in the system. To increase local knowledge available to leaders and thus the quality of the result, as well as the fraction of accepted invitations, we increased the number of links n per agent in t_2. However, despite better fitness, the number of refused invitations even increases slightly. We thereupon used only half the number of links to be rated by a leader to limit the number of refused invitations per leader and round to $m_r = 10$ instead of $m_r = 20$; we used $k_r = 5$ for all simulation runs where k_r is the maximum number of candidates a leader tries to extend its partition by. Compared to t_2, the fitness is still high but refused invitations are reduced by nearly 50%. Next, we varied the minimum reward τ_r necessary to become a candidate for integration. In t_4, we increased τ_r from -2 to 0 so that leaders do not send invitations to potentially non-beneficial agents. Compared to t_3, the number of refused invitations is reduced but, as $\tau_r = 0$ reduces the spread of local knowledge, fitness decreases. Regarding t_5, where τ_r is 1, the number of refused invitations is further decreased. The fitness for t_5 is low because, with respect to the composition of partitions, variety is too limited: sometimes, leaders could not find candidates for integration.

The reason for the strong increase of the fitness in the first rounds is that, at the beginning, the quality of the partitioning can quickly develop because many agents accepted the invitations. However, there are also very many refused invitations which indicates that local knowledge available to leaders was too restricted as partitions were small, or the minimal estimated reward necessary to become a candidate too low. After this short decline, the number of accepted and refused invitations dropped rapidly as some partitions terminated. Over time, active partitions reactivate some terminated partitions which leads to the temporal drop of the fitness, but increases the ability to rearrange memberships so that a higher fitness value could finally be achieved.

		F_{Avg}	F_{min}	F_{max}	T_{avg}	T_{min}	T_{max}
t_1	init	0.898	0.727	0.983	190.92	80	486
t_2	init	0.959	0.766	0.987	189.72	26	432
t_3	init	0.926	0.792	0.976	162.50	34	433
t_4	init	0.898	0.754	0.970	143.06	20	318
t_5	init	0.856	0.705	0.967	129.82	14	268
t_3	reconf	0.943	0.838	0.997	35.09	8	243
t_4	reconf	0.930	0.820	0.994	22.21	8	212
t_5	reconf	0.923	0.839	0.990	20.68	9	191

Table 4.2: Further Results for SPADA: Average Fitness when terminated, Minimal Fitness when terminated, Maximal Fitness when terminated, Average Termination, Minimal Termination, Maximal Termination

When comparing the results of tests t_1 and t_3, the fitness sightly decreases from 0.959 to 0.926. The number of refused invitations increases slightly, while the characteristic of the different graphs is maintained. As shown for t_2, a similar result can be achieved by decreasing the number of links per agent.

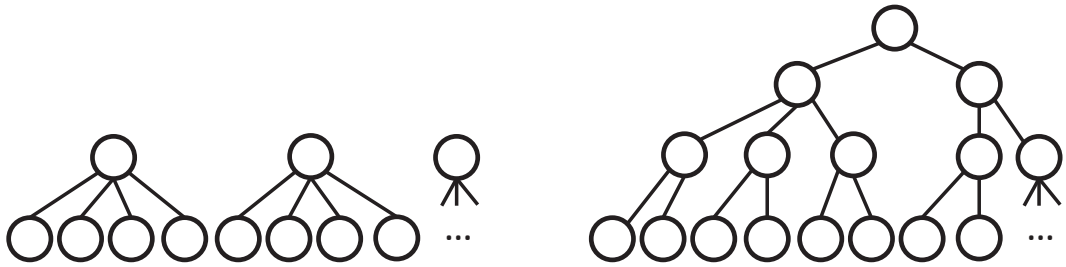
In addition to the tests for initial configuration, we evaluated the behaviour of SPADA for parameter sets t_3, t_4, and t_5 for reconfiguration, meaning that SPADA and PSOSP were initialized with a randomly generated partitioning consisting of 10 to 20 partitions of a size between 5 and 50 agents, and a rather high mean fitness of 0.875 as it might be the case in reconfiguration scenarios. Further, one random partition triggered reconfiguration while others were terminated. The results for t_3, t_4, and t_5 are similar to initial configuration (see Figure 4.4b and Table 4.2). Again, SPADA and PSOSP achieve a high fitness of 0.941 and 0.997, respectively. The number of accepted and refused invitations, however, are less than a tenth of the number of accepted and refused invitations when regarding initial configuration. However, we observed that SPADA sometimes can not complete reconfiguration for t_5 because of the high value of τ_r. In such a case, a leader can not identify a candidate for integration because its partition was situated in a local optimum from its perspective. Consequently, we identified t_3 as the best set of parameters. Although nearly all partitions were re-triggered over time, SPADA performs 119 changes on average for t_3 to reconfigure the system as most of the re-triggered partitions terminate after a few rounds, whereas it makes 1858 modifications to the initial partitioning. Hence, SPADA is very well-suited for reconfiguration where selective changes are often desired.

In all simulation runs, the undirected analogue of the acquaintances graph remained connected although its connectivity is not guaranteed by the current formulation of the algorithm. The reason for this is that it is rather unlikely to break the acquaintances graph into disconnected subgraphs if the number of links per agent is high enough. Summarizing, despite using local knowledge only, the quality of partitionings found by SPADA is within 10% of the solutions found by the centralised PSO. Moreover, SPADA allows selective changes which is beneficial in reconfiguration scenarios.

4.4 Autonomous Hierarchical Partitioning Control: HiSPADA

The original SPADA and PSOSP algorithms partition the set of agents representing the entire system. They create a flat hierarchy as shown in Figure 4.5a by solving the set partitioning problem on the agents in the system. To achieve hierarchical self-organisation as depicted in Figure 4.5 (b), HiSPADA (Steghöfer et al., 2013b) uses a set partitioning algorithm such as the original SPADA or PSOSP to partition only a subset of agents—the *neighbourhood*. The introduction and dissolution of layers is handled by the HiSPADA control loop, depicted in Figure 4.6. This control loop constitutes a *partitioning control*. It runs on those intermediaries that can be reorganised, i.e., that represent partitions that can be reorganised.

Thus, HiSPADA solves a *nested set partitioning problem*. It creates a partitioning on each hierarchy level with sets defined on the lower level as the elements to be partitioned. While in the SPP, the goal is to partition a set $\mathcal{A} = \{a_1, \ldots, a_n\}$ into $k \leq n$ pairwise disjoint subsets, in the nested SPP, the elements of A are themselves sets.



(a) Flat, single layer system structure. (b) Hierarchical system structure.

Figure 4.5: Different system structures. The flat, single layer system structure can be created with coalition formation, clustering, or a set partitioning algorithm such as SPADA while hierarchies can be introduced when using the HiSPADA control loop in combination with one of these mechanisms.

Prerequisites and Interaction with SPADA

In principle, the partitioning control is independent of the concrete set partitioning algorithm used. Therefore, it is be possible to use, e.g., a central formulation of the set partitioning problem such as the one introduced in Section 9.1 to find appropriate partitions and introduce hierarchies by using the partitioning control described here. HiSPADA's only requirements are (a) that it must be possible to limit the underlying algorithm to a certain neighbourhood and (b) that it must be possible to use application-specific formation and termination criteria to, e.g., define a minimal number of partitions to be formed. For reasons of conciseness, we will, however, focus our explanations on the control of SPADA and the PSOSP introduced above.

The hierarchy is represented by a tree-structure formed by father-child relationships between agents. To achieve this structure, an agent is introduced that serves as the root of the tree. When the hierarchy is updated, the relationships change. SPADA has a similar but distinct notion that expresses membership in a partition. As described in Section 4.1, each partition formed by SPADA has a leader that is at the root of a tree of marked links. HiSPADA makes no use of these structures used by the underlying set partitioning algorithm but only of intermediaries that represent the partitions. To form a hierarchy, these intermediaries are themselves partitioned. Thus, whenever an intermediary switches to another partition, the controlled agents switch with it. In the case study, an intermediary is always represented by an AVPP. Hierarchical structures originate from the introduction of new intermediaries on different levels of the existing hierarchy.

From a software and systems engineering point of view, it therefore makes sense to introduce concepts for micro-level system elements that can not be further broken down and higher, meso-level elements that can control others and can participate in the reorganisation. In the system of systems view used in this thesis, this corresponds to atomic system elements on the micro-level and systems of systems on the meso-level. In the context of the case study, the atomic elements are the individual power plants while the systems of systems are the intermediaries represented by AVPPs.

The HiSPADA control loop is usually dormant as long as the system is stable. It monitors the system however—more precisely: each instance of the control loop monitors the agent it runs on—and reacts to the violation of constraints. These constraints are depicted as guards on the transitions of the control loop in Figure 4.6. A run of SPADA and the introduction or dissolution of layers is thus an attempt to restore these constraints. This behaviour conforms to the Restore Invariant Approach, introduced in Section 5.1.

The HiSPADA control loop

HiSPADA has three major functional aspects: dissolve an existing layer of the hierarchy by removing intermediaries; introduce a new layer in the hierarchy by creating intermediary agents; reorganise a layer in the hierarchy by changing the relationships between intermediaries and the agents they control. In the following, we will use the term "hierarchy level" to denote all agents that are controlled by the same father or grandfather. Figure 4.6 shows the control flow of HiSPADA, including these three aspects.

In the power management case study, the dissolution and introduction of hierarchies is driven by performance constraints. As discussed earlier, the formation of structures in self-organising systems must aid the system in fulfilling its requirements. Hierarchical structures are ideally suited to compartment-

alise and thus increase scalability (see, e.g., (Horling and Lesser, 2004), Chapter 10 and Chapter 11). While a deep hierarchy ensures that the computational cost on each level are low, it prevents good solutions for the scheduling problem since these require degrees of freedom in each level. These degrees of freedom are provided by the controllable power plants in each AVPP. To achieve a fast and robust scheduling, it is necessary to make a trade-off between these factors. Therefore, hierarchy levels that schedule too fast—and thus have only a limited number of controllable power plants—are dissolved. The structuration within one level is driven by application-specific criteria that support the solution algorithms on that level. In the power management case study the structuration is driven by the mixture constraints introduced in Section 4.1.

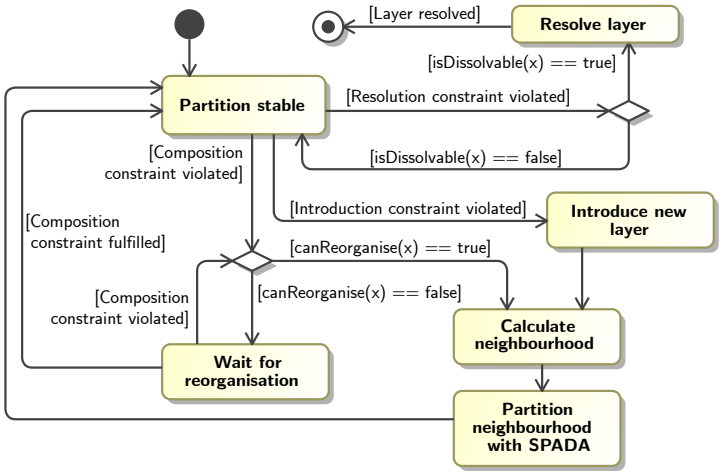


Figure 4.6: The HiSPADA control loop. The partitioning control running on intermediary x reacts to the violation of application-specific constraints, depicted here as guards, and reacts by dissolving layers of the hierarchy, introducing new ones, or reorganising existing ones.

Dissolve hierarchy levels The dissolution of existing hierarchy levels can occur for a number of application-specific reasons. In the power management example, it is triggered when the runtime of the scheduling algorithm falls below a given threshold. In that case, the AVPP that encountered this constraint violation is dissolved and all power plants are added to the father of the dissolved AVPP. In general, the predicate `isDissolvable`(x) is checked before the actual dissolution. It tests whether intermediary x can be dissolved at all.

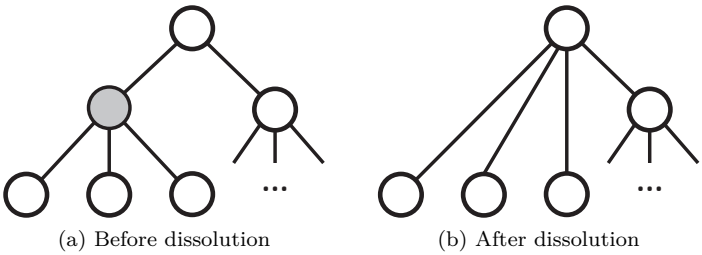
$$\texttt{isDissolvable}(x) \Leftrightarrow x.\texttt{mayBeDissolved} \wedge x.\texttt{timeInExistence} \geq \texttt{minTimeInExistence}$$

x.`mayBeDissolved` is false if intermediary x is a higher-level structure that is part of a predefined hierarchy. To avoid thrashing, it also checks if the period of grace (`minTimeInExistence`) that prevents newly formed hierarchy levels to be dissolved right away has already expired. The age of intermediary x is stored in x.`timeInExistence`.

Figure 4.7 shows a hierarchy in which a layer is dissolved. After the dissolution and before the initiating agent is deleted, it informs the new father agent of the changes made. The father agent then has to react appropriately by adopting its new children and by, e.g., requesting essential data from them or running the control algorithm again. In the decentralised power management case study, all information necessary to integrate the children into the scheduling performed by their new father must be transmitted. The childrens' current schedules remain valid and can be maintained until the father creates new ones.

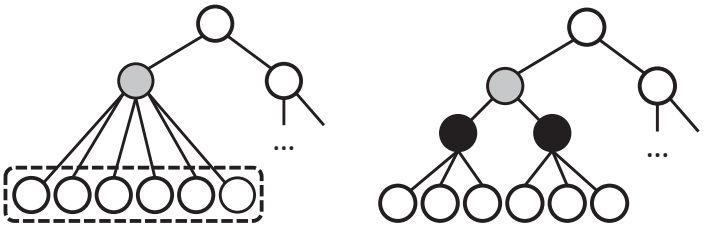
Introduce new hierarchy level In the case study, new intermediaries and thus new hierarchy levels are introduced when an AVPP requires too much time to calculate the schedule for the power plants it controls. Other applications can of course give specific conditions under which this action is performed.

When a new hierarchy level is introduced, a father agent f creates an intermediary level for its child agents. For this purpose, f's HiSPADA control loop initialises SPADA with its child agents as the neighbourhood and a minimum number of two partitions, thus ensuring that the agents are not subsumed in just one partition. A way to guarantee this is to require that each leader knows at least one other leader. SPADA or PSOSP then use this and other, application-defined criteria to create a



(a) Before dissolution (b) After dissolution

Figure 4.7: Dissolution of a hierarchy level. The initiating agent is marked in grey. Children of the initiator become children of their previous grandfather. After the dissolution the previous grandfather has to react to the new children accordingly, e.g., by performing the control algorithm again.



(a) Before reorganisation. (b) After introduction of new layer.

Figure 4.8: Introduction of a new hierarchy level. The initiating agent is marked in grey, new agents are black. The child agents of the initiating agent form the neighbourhood, marked as a dashed rectangle, that is partitioned with the set partitioning algorithm. New agents are introduced to form an intermediary layer.

suitable partitioning as detailed in "Reorganising a hierarchy level" below. Figure 4.8 illustrates this process. The newly created agents become children of f.

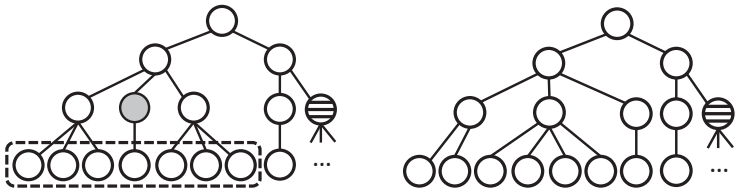
In the power management system, the introduction of a new intermediary means that the schedules of the children of the new intermediary are still valid. Since the intermediary does not produce any additional power, the power plants assigned to it can stick to their current schedule and the intermediary's father only has to subsume the load previously assigned to its children under the new intermediary. Subsequent scheduling runs, however, have to take the new structure into account. Therefore, the new intermediary has to gather data from its children and prepare the scheduling problem as described in Chapter 10 and Chapter 11.

Reorganising a hierarchy level Whenever intermediary x detects violation of an application-specific composition constraints, x uses HiSPADA to reorganise a hierarchy level. For this purpose, it limits the scope of reorganisation to a certain neighbourhood, thus preventing it from crossing organisational boundaries. The original SPADA and PSOSP have no such limitation and reorganise the entire system by default. The application-specific constraints that lead to a reorganisation of a hierarchy level in the case of the decentralised power management case study are the mixture of credibility and energy source between the AVPPs on the same level. As introduced in the discussion of SPADA, a good mixture supports the creation of schedules for the AVPPs.

HiSPADA has to consider some limitations when a hierarchy level has to be reorganised. First of all, reorganisation can not occur while an agent's father or children are being reorganised. Otherwise it would be possible that some agents are part of several reorganisation efforts at once, possibly resulting in changes that would violate the tree-structure of the hierarchy. This limitation is captured in the $\texttt{canReorganise}(x)$ predicate that is tested before reorganisation occurs. The predicate $\texttt{father}(p, q)$ denotes that p is the direct predecessor of q in the hierarchy. If an agent p currently is reorganising or is being reorganised, $\texttt{reorganising}(p)$ evaluates to $\texttt{true}$.

$$\begin{aligned}\texttt{canReorganise}(x) \Leftrightarrow\ & \texttt{isDissolvable}(x) \\ & \wedge \neg\exists y \in \mathit{Agents} : \texttt{father}(x, y) \wedge \texttt{reorganising}(y) \\ & \wedge \neg\exists z \in \mathit{Agents} : \texttt{father}(z, x) \wedge \texttt{reorganising}(z)\end{aligned}$$

The second limitation is the restriction of the algorithm to neighbourhoods. In this case, the neighbourhood of the initiating agent is defined as all its children and "nephews", i.e., its siblings' children (cf. Figure 4.9). Therefore, we define the neighbourhood N_x of an agent x as follows:



(a) Before reorganisation. (b) After reorganisation.

Figure 4.9: Reorganisation of a hierarchy level. The initiating agent (corresponding to intermediary x) is marked in grey and the neighbourhood is marked by a dashed rectangle. The dashed agent can not be reconfigured.

$$N_x := \{y \in \mathit{Agents} \mid \exists a, a_y \in \mathit{Agents} : \\ \texttt{father}(a, x) \wedge \texttt{father}(a, a_y) \wedge \\ \texttt{father}(a_y, y) \wedge \texttt{canReorganise}(a_y)\}$$

This neighbourhood definition ensures that only agents with fathers that can be reorganised are part of the neighbourhood, thus ensuring that predefined hierarchies or levels that are still in their period of grace are not changed. In theory, neighbourhood definitions that include more distant relatives are possible and can be implemented easily in HiSPADA. However, recursing the hierarchy up too far reduces the benefit as including more agents in the neighbourhood makes the partitioning problem more complicated and agents that are only distantly related to each other have usually been separated by HiSPADA in the course of hierarchy creation. Partitioning them closer together is not useful and might even jeopardise the system goal. Also, this approach ensures that HiSPADA provides a local solution with only a limited amount of communication.

After the neighbourhood has been determined, the partitioning algorithm is initialised with the set of agents in N_x and a minimal number of new partitions. Then, the partitioning algorithm is executed the same way as in the non-hierarchical case. After the algorithm has been run, existing intermediaries are reused or—depending on the number of created partitions—new ones are created or old ones removed. Figure 4.9 shows an example for the reorganisation of a hierarchy level.

Bootstrapping the system HiSPADA assumes very little about the initial conditions of the system. As the partitioning control runs on the agents of the system, a hierarchy will develop in the system if the constraints monitored by the control loop are violated. If there is no hierarchical structure to begin with, an initial run of SPADA can establish a partitioning on all agents in the system. This ensures that the initial partitions are suitable for the purposes of the system as SPADA uses application-specific metrics in the process. The (flat) hierarchy introduced by SPADA or PSOSP can easily be transformed into a tree by establishing a root agent that subsumes the intermediaries representing the newly formed partitions. This root is not dissolvable and stays at the root of the hierarchy throughout the runtime of the system. Instead of running SPADA or PSOSP, the system can also be partitioned randomly. In such a case, however, HiSPADA will take a while to find a suitable hierarchy if the initial partitioning did not make use of application-specific criteria. As this constitutes a worst case scenario, it is used in the evaluation.

If a pre-defined organisational structure exists, corresponding partitions have to be initialised. These agents have appropriate relationships with their children to depict the organisation. HiSPADA can then work on this hierarchy by introducing intermediaries and dissolving hierarchy layers formed by those intermediaries. It is therefore possible to let the partitioning control find a suitable sub-hierarchy for each of the predefined organisational entities. Of course, there is a trade-off between the fine-grained depiction of existing structures and the organisational prowess of HiSPADA: if the predefined structure is too rigid, the partitioning control is not able to tackle the scalability issues it is intended for.

4.5 Performance of HiSPADA in the Decentralised Power Management Case Study

Evaluating a hierarchical self-organisation algorithm makes it necessary to define the environment in which it operates. As discussed above, HiSPADA operates according to application-specific constraints and the underlying partitioning algorithms operate according to application-specific fitness functions.

	Sc. A	Sc. B	Sc. C	Sc. D
Max. sequential runtime of scheduling in ms	3624 ± 55	925 ± 638	499 ± 220	484 ± 213
Avg. height of hierarchy	–	2	4.99	3.52

Table 4.3: Initial evaluation results for scalability for a system consisting of 435 power plants from (Steghöfer et al., 2013b). In scenario A, no hierarchies exist while the original SPADA created a flat hierarchy in Scenario B. HiSPADA can work without predefined organisations (Scenario C) and within a predefined hierarchy, here with an initial height of 3 (Scenario D). We define the height of the hierarchy as the length of the longest downward path to a leaf from the root.

Therefore, we evaluate HiSPADA in the context of the decentralised power management case study, in particular w.r.t. the resulting hierarchies ability to solve the scheduling problem effectively. The evaluations in (Steghöfer et al., 2013b) showed that HiSPADA is able to reduce the *average maximal sequential scheduling times* significantly. This key measure results from traversing the hierarchy from the top-level AVPP to the leaves, summing the scheduling times over all nodes in a distinct path. Since scheduling is a sequential process, i.e., the top-level AVPP creates the schedules for its children first, then the children create the schedules for their children and so on, selecting the maximum resulting scheduling time is a kind of worst case estimation for the current structure. Since the results are averaged over a number of runs, they are representative of different structures and situations. These preliminary results are shown in Table 4.3.

The dependency between HiSPADA and the scheduling mechanism makes it hard to find suitable parameter sets for HiSPADA. Especially as different solution approaches to the power plant scheduling problem can have vastly different runtimes (cf. Chapter 9), HiSPADA is sensitive to changes in the scheduling process, especially if the runtime bounds for the scheduling process are set rather tightly. In addition, the possible number of combinations of parameters for SPADA, HiSPADA and the scheduling algorithm is enormous. The following evaluations are therefore performed with sets of parameters that are representatives of different "families" of settings. As the individual "families" can differ quite tremendously, the differences between the results for the families can give an indication of the influence the parameters have on stability and performance of the system. The results are therefore presented in a comparative way.

Evaluation Settings and Parameter Families

To give a baseline for evaluations, a simple scenario without any hierarchies is used. It has a "super-flat" hierarchy, i.e., a grand coalition consisting of one top-level AVPP that controls all 523 power plants used in this scenario and all others. No self-organisation is present and the AVPP performs scheduling for all of the 173 controllable power plants. All power plant data is based on real power plants in the administrative district of Swabia, Bavaria from freely available resources[4]. Of special interest are the scheduling times, achieved with a simple centralised model[5] that does not allow switching power plants on or off and does not take into account individual optimisation criteria. The schedules are optimised for a low violation, i.e., a minimisation of the difference between production and consumption.

The main difference in the scenarios—corresponding to the parameter "families"—used for the evaluations are the initial hierarchy, the sensitivity of the application-specific constraints—i.e., the allowed range for the scheduling runtimes—and how strictly the constraints are enforced. Table 4.4 shows the different settings. Especially the last point is interesting since it adds a degree of leniency into the mix: if the scheduling runtime is violated once, nothing happens, but if it is violated twice within four time steps, a reorganisation is triggered. These MaxSPAN constraints are discussed in more detail in Section 6.2. Here, the effect is to reduce volatility in the results that stems from occasional fluctuations in the scheduling times. In addition, the trigger for the reorganisation of a level—the difference in the credibility mix—is an important factor.

The setting was the same for all scenarios: after an initial bootstrapping phase of 32 time steps in which the system collected data about the environment and formulated predictions about the behaviour of the demand and the intermittent power plants, the self-organisation algorithm started to evaluate

[4] `http://energymap.info/energieregionen/DE/105/111/169.html`, power plant data from May 2012.

[5] See Chapter 9 for a discussion of different scheduling models and their complexity.

	Baseline	Family A	Family B	Family C	Family D
Initial hierarchy height	1	2	3	3	3
Minimal/Maximal scheduling time	N/A	30, 200	50,120	50, 120	30,200
Allow x violations in y time steps (x, y)	N/A	1,1	1,1	2,4	2,4
Allowable difference in credibility for children	0.1	0.1	0.1	0.1	0.1

Table 4.4: Evaluation parameters defining the different "families" of parameter settings. Family A: Flat Hierarchy, Lenient Scheduling; Family B: Deep Hierarchy, Strict Scheduling; Family C: Deep Hierarchy, Strict Scheduling with MaxSPAN constraint; Family D: Deep Hierarchy, Lenient Scheduling using a MaxSPAN constraint. The allowable difference in credibility for children of an AVPP is set to a fixed value as a result of the way the system treats credibility values. It ensures that the initial mix of the credibility is insufficient but that a stable structure can be found eventually. Evaluations with MaxSPAN constraints allow a maximum of two violations within four time steps.

the constraints and thus to change the hierarchy. In time step 80, the behaviour of the stochastic power plants changed and their systematic error was altered. This requires the scheduling system to alter its internal knowledge about the agents and thus causes the scheduling times to rise temporarily. Such a destabilising factor gives an indication of how sensitive the hierarchy is to changes in the environment and whether or not it is possible to react adequately. The influence it has on the scheduling times in the baseline case are evident in Figure 4.10. Schedules are created based on credibility-based scenarios (cf. Section 9.3) for a predicted demand.

The number of experiments and the concrete settings for the different evaluation runs are summarised in Table 4.5. At least 100 experiments per run were required for statistically significant conclusions. Some runs have less than this number but are only used for comparison and to show differences to the standards that are obvious with single incidences, e.g., for the use of the particle swarm optimiser or for larger numbers of agents.

Parameters	Algorithm	No. of agents	No. of experiments
Baseline	N/A	523	120
Family A	SPADA	523	140
	SPADA	1000	12
Family B	SPADA	523	112
Family C	SPADA	523	112
Family D	SPADA	523	120
	PSO	523	14

Table 4.5: Overview of evaluation runs for the different parameter families, indicating the used set partitioning algorithms, the number of agents, and the number of runs used to obtain the data in the discussion of the evaluations and the figures.

SPADA has been parametrised as in test t_3 discussed in Section 4.3, with the sole exception that the number of links per agent was set to 12 instead of 20 to increase SPADA's performance and reduce the communication overhead. The quality of the partitioning is, however, very similar to the one reported for t_3. To ensure termination, SPADA is only allowed to run for a maximum of 7 rounds.

Evaluation Hypotheses and Results

To structure the evaluation and its results, a number of hypotheses were formulated and tested. They were formulated so as to be independent of the algorithm HiSPADA is combined with. Since the results of the hierarchy formation depend critically on the properties of the algorithm HiSPADA is coupled with, however, using the approach in a different setting will make it necessary to re-evaluate the parameters and the interference of the algorithms. The results shown below can be a guideline and illuminate the most important aspects of the interplay between HiSPADA and a concrete application.

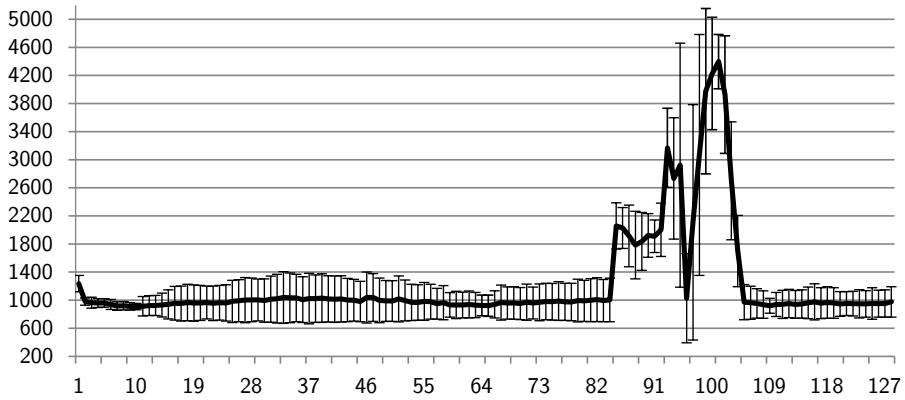


Figure 4.10: The average scheduling times of the baseline scenario over a run of 128 time steps. Up until the change in behaviour, the scheduling runtimes are relatively constant at about 1 second per time step but increase dramatically for about 25 time steps after the change. The standard deviation also rises accordingly and stabilises again afterwards.

The expectations for the different parameter families are relatively clear. Families with a greater range of acceptable scheduling times should create more stable hierarchies that might, however, not be as optimal when it comes to scheduling times. Using MaxSPAN constraints should further this stability since temporal fluctuations will not cause an immediate reorganisation. Finally, the initial height of the hierarchy should alter the behaviour of the system in the beginning but regardless of initial height, a similar hierarchy should evolve over time.

The important features of the hierarchy formation are said similarity, the average sequential runtime within the system, the credibility mix, and the stability of the hierarchy. Similarity of the hierarchies can be measured by comparing the height and the average number of children per AVPP. The average sequential runtime used here is defined as the average of the sums of the sequential scheduling runtime on all paths in the hierarchy. It is used instead of the maximum sequential runtime in the preliminary evaluations since peculiarities of the current scheduling process lead to rather high and unrepresentative standard deviations. The credibility mix gives an indication of the difference between the AVPPs in terms of the power plants they control. A low credibility value indicates that an AVPP has to deal with untrustworthy power plants. Since it is the goal to distribute untrustworthy agents in the system, the difference in credibility values—i.e., the credibility mix deviation—should be as low as possible. Finally, stability of the hierarchy can be measured by the number of active partitions in each time step.

Hypothesis: Average maximum sequential runtime is reduced with HiSPADA This hypothesis was the object of investigation in the original evaluation presented above. The numbers in Table 4.3 show that the version of the algorithm tested there indeed reduced the average maximum sequential runtime significantly. These finding have been substantiated in the new evaluations. The average sequential runtime of scheduling is reduced from about one second to less than 100 ms. The impact of the change in behaviour can still be seen, especially because of the standard deviation it incurs. Overall, the primary purpose of HiSPADA is thus achieved.

The baseline scenario exhibits scheduling runtimes as depicted in Figure 4.10. They are relatively stable until the change in behaviour occurs in time step 80 that lead to temporarily higher scheduling times until a stable state is reached again. The other scenarios should exhibit a similar curve, albeit we expect that the scheduling runtimes to be lower to begin with since the system starts with a form of hierarchical structure in which the work can already be distributed. We further expect scheduling times to sink after the initial structural reorganisation finishes and the system had a chance to optimise. The disturbance in time step 80 should then lead to a similar pattern as the one shown in Figure 4.10.

As Figure 4.11 shows, the expected reduction actually occurs. Indeed, it is quite significant and similar to the reductions found in the original evaluation. More interesting are the outliers apparent in the graph. They are due to a peculiarity of the version of the scheduling algorithm the evaluation was performed with. Under some circumstances, solution times were much higher than normal. This also explains the drastic variation in the standard deviation. For parameter family A, e.g., standard deviations go up to more than 20 seconds after the change in behaviour occured. Such drastic fluctuations in the scheduling time are, of course, a challenge for HiSPADA and the stability of the hierarchies. Indeed,

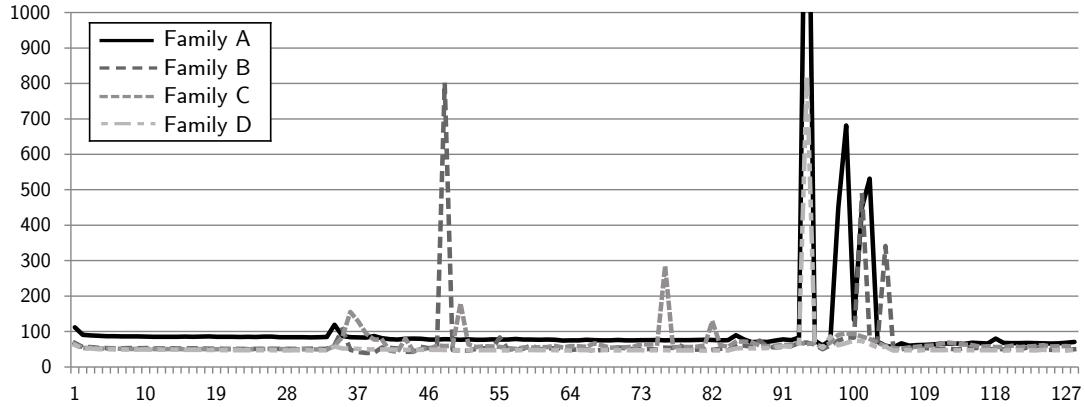


Figure 4.11: Average scheduling times for the different parameter families. Family A has a peak scheduling time of 1857.2425ms due to issues with the scheduling algorithm described in the text. Overall, a significant reduction in scheduling time is evident when compared to the baseline.

activity increases in the steps subsequent to these anomalies. However, using MaxSPAN constraints is a viable way of dealing with such outliers and promoting stability.

Interestingly, parameter families with a strict scheduling time range do not show significant differences in scheduling times, even after the disturbance in the system. A side-by-side comparison of family A and family B in Figure 4.12 show this phenomenon: Even though family B—the one with the stricter settings—has slightly lower scheduling times, the standard deviations are higher and thus both solutions are relatively close to each other. This is possible due to the fact that the scheduling times in the system generally cluster around a value of 65 to 70 ms as witnessed by the initial scheduling times for the time steps before HiSPADA kicks in as soon as a reasonable hierarchy exists. The variant of the scenario using parameter family A with 1000 power plants shows the reduction in scheduling time that follows the restructuration with HiSPADA as depicted in Figure 4.13.

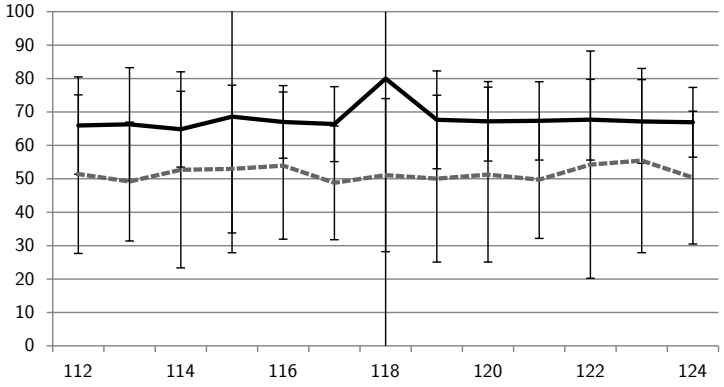


Figure 4.12: A comparison of scheduling times for parameter families A and B after the disturbance of the system has been dealt with. Family A exhibits slightly higher scheduling times but the standard deviation of the scheduling times for family B overlap A's values.

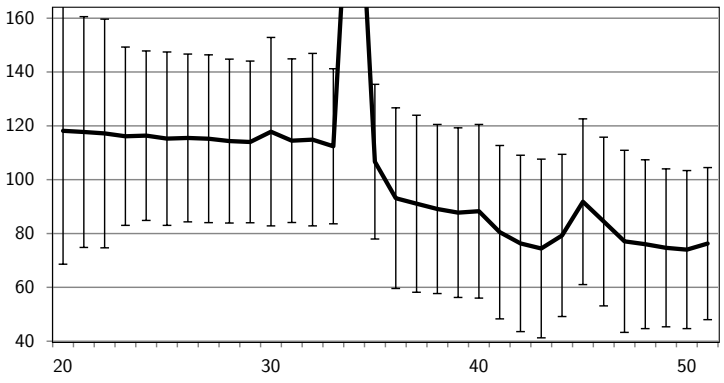
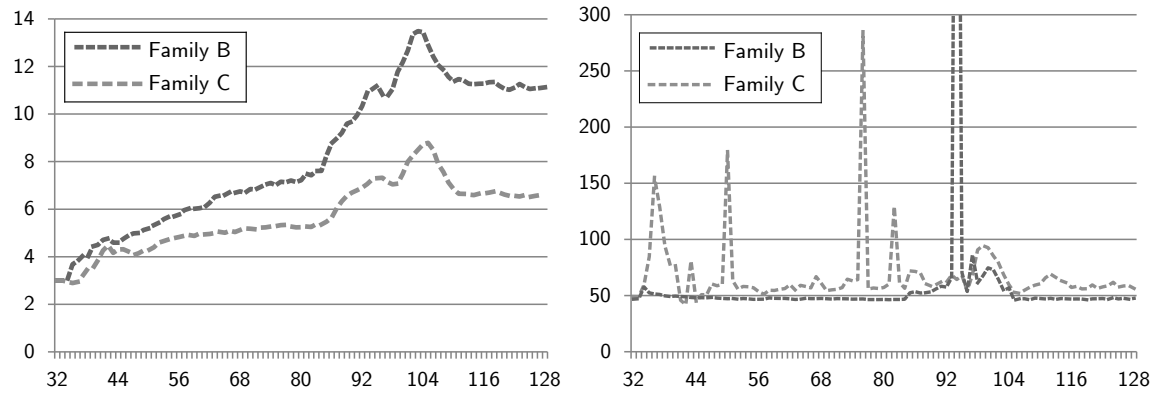


Figure 4.13: Scheduling times in for the variant of parameter family A with 1000 power plants. Since each AVPP in the hierarchy initially has to create schedules for more power plants, the impact of the hierarchical self-organisation becomes evident. Apart from the outlier in time step 34 that can be considered an artefact, scheduling times decrease after HiSPADA has been activated due to the introduction of additional intermediaries.



(a) System structure height.

(b) Average sequential scheduling times.

Figure 4.14: Comparison of the development of hierarchy height and of the scheduling times for parameter families B and C. Both families use rather strict constraints on scheduling times but family C uses a MaxSPAN constraint to avoid reorganisations due to singular events. It has a lower hierarchy but also allows occasional outliers in scheduling times. The overall scheduling time, however, is only insignificantly higher than that of family B.

Hypothesis: Using MaxSPAN constraints and/or lenient scheduling times reduces reorganisations but increases overall scheduling times. Since more lenient constraints or the ability to overlook singular constraint violations with the use of MaxSPAN constraints reduces the number of times HiSPADA kicks in to reorganise the hierarchy, the system should be flatter and at the same time scheduling times should be slightly higher in these settings.

This hypothesis is supported by the data shown in Figure 4.14 and from Figure 4.15. The former figure shows that parameter family C—using MaxSPAN constraints—creates a more shallow hierarchy while the scheduling times are slightly higher than for parameter family B. The number of reorganising partitions is significantly lower as depicted in the latter figure. Both families still have relatively many active partitions compared to the other settings but this is due to the relatively tight corridor for scheduling. Family C has, in any case, a significantly lower number of reorganising partitions over the entire lifetime of the system.

Hypothesis: HiSPADA supports the creation of stable hierarchies. After HiSPADA has found a suitable partitioning, this structure provides a good basis for schedule creation and does not change dramatically any more unless the system is disturbed. This should be evident in all scenarios. After HiSPADA is activated in time step 32, an initial phase of reconfigurations should start that changes the structure more or less drastically in the beginning. After a while, most of the reorganisations should terminate and a phase of relative stability should be observable. When the system is disturbed in time step 80, however, the system structure should change quite dramatically before stabilising again in subsequent time steps.

Figure 4.15 and Figure 4.16 show the hypothesized behaviour for all paramter families. The graphs show that HiSPADA reacts to the current situation by starting a number of reorganisations in different partitions and how the system then stabilises after the partitions are again within the parameters set by the constraints. The stricter settings of family B and C along with the volatility of the scheduling algorithm cause a higher fluctuation than the other parameter families, but overall, the stabilisation especially after the disturbance event is clearly visible. The detail in Figure 4.14 is worth another look. After the disturbance occurs, the strict constraints cause the height of the hierarchy to increase in response to the rise of the scheduling times. When those stabilise again, the system dissolves a number of hierarchy levels and then enters a phase of stability with a constant number of levels.

Hypothesis: HiSPADA can be used for the initial hierarchical structuration of the system and in the context of existing hierarchical structures. If no or little initial structure is given, i.e., only a shallow hierarchy is provided so that a small number of AVPPs are directly assigned to the top-level AVPP, HiSPADA can organise the system appropriately. If a structure exists, HiSPADA can adapt this structure to suit the requirements of the application.

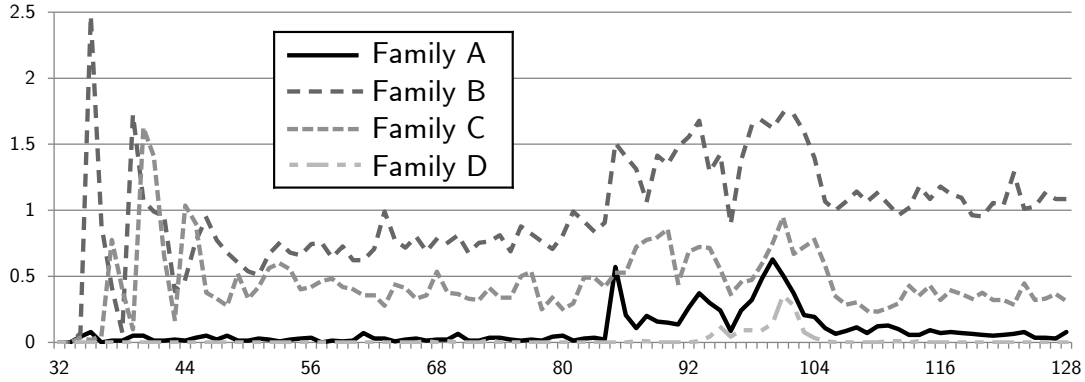


Figure 4.15: Mean number of reorganising partitions for the different parameter families. Although it can be seen that the activity and thus the change in the system depends mainly on the sensitivity of the constraints that guide the self-organisation process, the hypothesis holds and after an initial phase of reorganisations the system stabilises before reacting to the disturbance and then stabilising again.

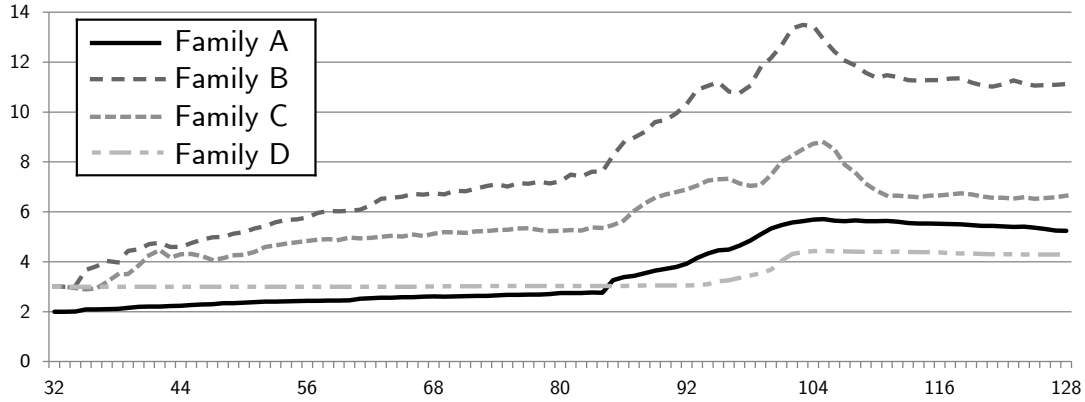
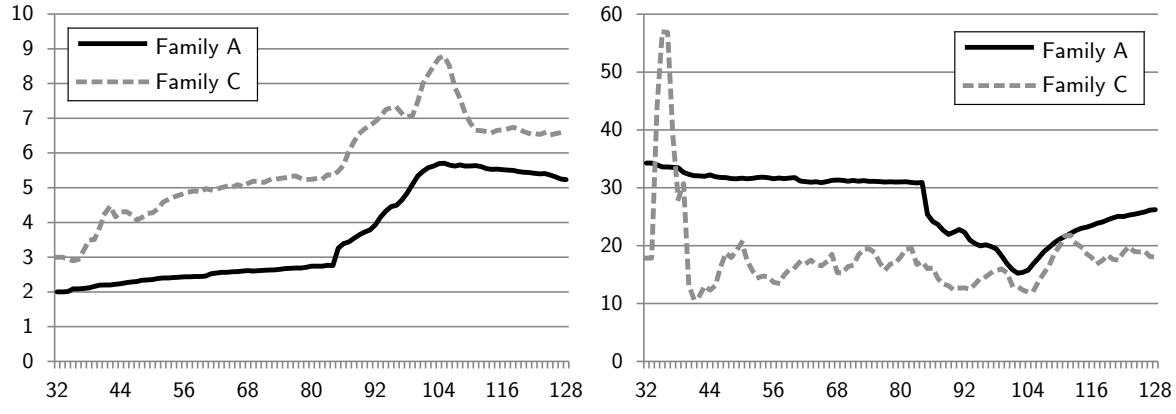


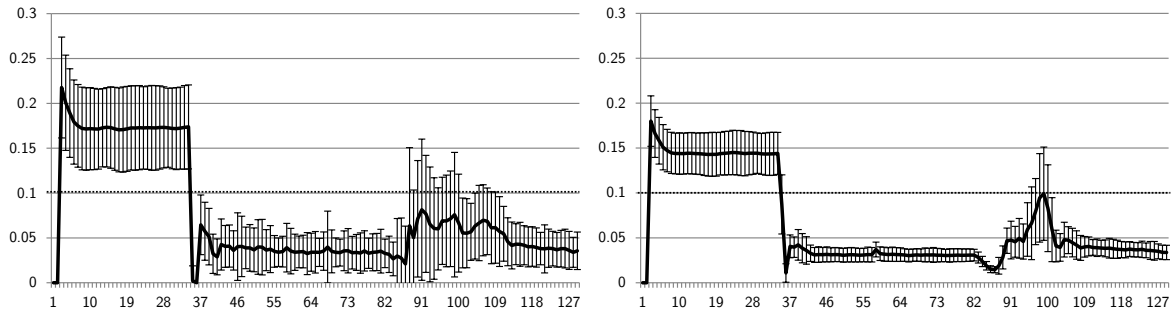
Figure 4.16: Mean hierarchy height for the different parameter families. In all scenarios, the hierarchy stabilises after the initial reorganisation phase and after the disturbance have been dealt with. This reflects with the number of active partitions in Figure 4.15 which decreases as the hierarchy height becomes stable in the aftermath of these events.



(a) System structure height.

(b) Number of children per AVPP.

Figure 4.17: Comparison of the development of mean hierarchy height and mean number of children per AVPP after HiSPADA has been activated for parameter families A and C. Family A starts with a flat hierarchy while family C uses an initial deep hierarchy. The results indicate that starting with a deep hierarchy promotes a deep hierarchy over the system lifetime while starting with a flat one leaves the system relatively flat.



(a) Parameter Family A. (b) Parameter Family D.

Figure 4.18: Development of mean credibility mix deviations over time for different parameter families. Both charts show a dip in the mix deviation shortly after time step 32 when the hierarchical self-organisation has been activated. This is due to the fact that at this point new hierarchy levels are introduced and existing ones are reorganised. At this point in time, some AVPP have no credibility value yet and others have not updated theirs. The change in behaviour is visible after time step 80 but has no drastic influence on the intra-level reorganisation. After the initial change in structure, the credibility mix is relatively stable and does not cause constraint violations.

As the scheduling times in Figure 4.11 indicate, HiSPADA is indeed able to find a suitable system structure starting from both flat and deep hierarchies. The influence of the initial configuration on the height of the hierarchy and the number of children per AVPP is another matter, however. The development of these figures for family A that uses a flat initial structure and family C that uses a deep one is shown in Figure 4.17. Over the entire lifecycle of the system, family A maintains a lower height and a higher number of children per AVPP. The deeper configuration introduces more layers and has a significantly lower number of agents per AVPP. The initial rise in children per AVPP is potentially due to the fact that many small AVPPs are dissolved when HiSPADA is activated due to violations of the minimal scheduling time. This does not necessarily affect the hierarchy height since it is possible that other AVPPs survive this purge. Dissolving multiple AVPPs leads to the creation of relatively large AVPPs and the power plants in these new large AVPPs are subsequently distributed in the hierarchy by further reconfigurations.

Hypothesis: HiSPADA finds a suitable organisation within a hierarchy level Apart from the fact that HiSPADA introduces and dissolves hierarchy levels, it also optimises the structure within one level, i.e., within the children of an AVPP. In the case study, the quality of this internal structure is determined by the credibility mix deviation. This value measures how different the AVPPs are with regard to the credibility values of their subordinates. Ideally, the untrustworthy power plants are distributed among the AVPPs so that each AVPP has to deal with approximately the same amount of uncertainty.

As Figure 4.18 shows, HiSPADA quickly structures the system in a way that the credibility is well mixed within a level. After time step 32, when HiSPADA is activated, the reorganisation immediately changes the structure within one level so that the constraint is not violated subsequently. Introduction and dissolution of levels also does not disturb this balance since SPADA—and the PSO as well—considers the credibility mix in its decision making.

Hypothesis: HiSPADA finds a stable compromise between hierarchy depth and scheduling times eventually. This hypothesis can be tested by checking for eventual termination of HiSPADA. If no more constraints are violated, i.e., scheduling takes neither too long nor is it too quick, the hierarchical self-organisation stops. The number of active partitions becomes 0 and the number of AVPPs as well as the hierarchy depth stays constant. Since all experiments are influenced by random environmental factors, the evaluations do not always show this behaviour. However, it can be seen that the overall trend supports this claim.

Supporting data for this hypothesis can again be found in Figure 4.11—depicting the mean scheduling time of the different parameter families—and Figure 4.16, depicting the mean hierarchy height. Before the system disturbance, scheduling times are relatively stable and the hierarchy, although in-

creasing slightly for the stricter parameter settings is rather stable as well. The disturbance then causes both the scheduling times and the hierarchy height to spike with a slight offset as would be expected. After the disturbance, both the scheduling times and the hierarchy height stabilise again, even though the hierarchy remains a little higher than before.

Hypothesis: HiSPADA can find suitable hierarchies within predefined hierarchical structures In many cases, some hierarchical structure will be pre-defined, e.g., in case existing companies are represented with their own AVPPs. HiSPADA is still able to find appropriate structures that respect pre-defined organisations. This is intuitively clear if the pre-defined AVPP is considered the top-level AVPP of an isolated sub-system. HiSPADA then works in this compartment as it would on any other system and its ability to find appropriate organisational structures is applied within that compartment.

Interpretation and Conclusions

In summary, the hypotheses put forward about the behaviour of HiSPADA have been substantiated by the data collected for the evaluations. However, a number of interesting aspects have come to light that merit further discussion.

Existence of *any* hierarchy is useful to reduce scheduling times. The creation of schedules for the 173 controllable power plants contained in the 523 power plant scenario takes about one second in the baseline case. As the initial 32 time steps in Figure 4.11 show, the existence of *any* hierarchical structure reduces this scheduling time significantly. While this is due to the fact that the divide-and-conquer approach is independent of the way the hierarchy comes about, it does not allow making any assumption about the quality of the schedules created. Indeed, the quality depends on the credibility mix, since it is beneficial to partition untrustworthy and trustworthy agents together to locally deal with uncertainties (Anders et al., 2013a). As Figure 4.18 shows, HiSPADA is able to reduce the differences between AVPPs in this regard and thus improve the quality of the structure in support of the scheduling process.

Trigger-happy parameter settings do not provide a good compromise between scheduling time and hierarchy height. Parameter families B and C create rather high hierarchies due to the fact that the constraints define a very tight corridor for the scheduling times as shown in Figure 4.14a. Family B in particular does not tolerate any deviation from the tight bounds for scheduling times and creates a very high hierarchy that violates the principle of compromising between controllability and complexity. At the same time, as witnessed by Figure 4.11, the scheduling times in these hierarchies are not significantly better than in other, less obstructive parameter families.

The number of power plants is not a relevant factor for HiSPADA. Even though the number of experiments for settings of 1000 agents was relatively low and did not yield statistically significant data, the results indicate that HiSPADA works with large systems and achieves its goals in such a setting. This is not surprising since HiSPADA always works in a localised fashion on individual partitions and only with the agents directly controlled by one intermediary. Therefore, each instance of HiSPADA and of the set partitioning algorithm controlled by it only controls a limited pool of agents that does not increase with a higher overall number of agents in the system. In fact, the pool size is not limited by HiSPADA but by the employed set partitioning algorithm. The evaluations in Section 4.3 show that HiSPADA is able to deal with systems with several hundred agents given enough time. HiSPADA's communication is also limited within the hierarchy and a single instance only communicates locally. Therefore, the combination between HiSPADA and SPADA is expected to have the ability to scale almost indefinitely as long as the individual pool of agents is initially bounded.

SPADA finds solutions that are in some respects better than those of the particle swarm optimiser Although the number of runs with the PSO is also too small to make statistically significant assertions, the available data points towards the impression that the credibility mix deviation (the decisive factor in the calculation of the fitness value in Section 4.3) is handled better by SPADA than by the PSO as shown in Figure 4.19. This phenomenon might be attributable to the way the PSO jumps through the solution space. A more in-depth investigation is pending. However, since the PSO's solutions are still within the boundary of the constraints, no additional reorganisations are triggered in response to these solutions.

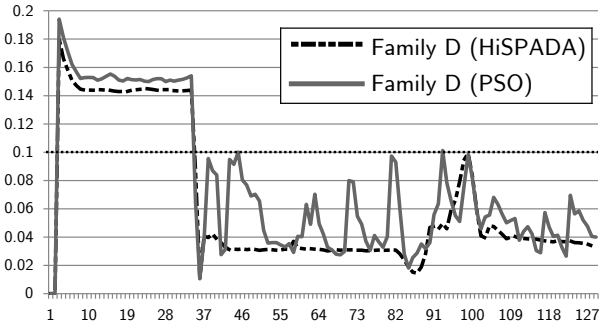


Figure 4.19: Comparison of credibility mix deviations for solutions of parameter familiy D with HiSPADA and the jumping frog PSO described in Section 4.2. The PSO shows worse solutions for the credibility mix than SPADA for almost all runs, although staying below the threshold defined by the credibility mix deviation constraint.

4.6 Comparison to Hierarchical Self-Organisation from the Literature

Hierarchical clustering algorithms are, e.g., regarded in sensor networks. There, groups of sensors represented by a cluster head are formed. The head serves as a communication hub for the entire group. The number of hops (nodes a message has to go through to reach its destination) and the communication range are vital decision variables. Additionally, as the network's structure can change arbitrarily at runtime and robustness is of great concern, centralised approaches are not employable. As (Bandyopadhyay and Coyle, 2003) show, hierarchies can significantly reduce the complexity of cluster formation. This result also shows that hierarchies increase scalability in complex systems.

In sensor networks, highly decentralised algorithms such as ACE (Chan and Perrig, 2004) or HEED (Younis and Fahmy, 2004) are used for clustering. Likewise, the LEACH algorithm (Heinzelman et al., 2000) creates a flat hierarchy of clusters, led by randomly selected cluster-heads. The sensors decide which cluster they belong to by selecting the cluster-head that requires the least radio strength to reach. To avoid draining the battery of cluster-heads too quickly, the algorithm rotates the role of cluster head among all capable sensors. Unfortunately, results and algorithms from sensor networks can not be readily applied to other domains as they work under specific assumptions. Most importantly, they frequently use radio strength as a guiding principle of cluster formation. Conversely, it will be difficult to adapt HiSPADA to sensor networks as it does not account for issues such as energy efficiency or limited communication.

The formation of hierarchical structures can also follow the hierarchical decomposition of the task the agent system has to fulfil. The domain of sensor networks is represented here again in the work of Kinnebrew and Biswas (2009). The authors propose a hierarchical contract net protocol that allows groups of agents to bid for the assignment of a sub-task. The group can then decompose the sub-task into smaller work units and use the same protocol again. Thus, not only the work is distributed but also hierarchically structured organisations emerge. As observed by Horling and Lesser (2004), the original formulation of the contract net protocol (Smith, 1980)—which has not been intended for sensor networks—was aimed at creating long-lasting organisational structures through hierarchical distribution of sub-tasks, although this notion has later been dropped in favour of short-time contract fulfilment.

More generally, coalition formation algorithms or task allocation algorithms (Gerkey and Matarić, 2004) solve the set partitioning problem—and thus the clustering problem—during the process of *coalition structure generation*. Specialized algorithms for this task are often not fully distributed as, e.g., global knowledge is required to calculate the search space (Rahwan et al., 2009). Even if the search space (i.e., the set of all possible coalitions) is distributed (Shehory and Kraus, 1998), such an approach still requires a lot of communication, scales badly, and is not applicable in systems with a fluctuating agent population. In other cases, pre-defined organisational structures are exploited to guide the search for coalitions. This enables the use of local knowledge and neighbourhood relations. While the work of Abdallah and Lesser (2004) is based on a hierarchical system structure, Anders et al. (2011a) require a graph structure that defines input/output relations between agents.

SPADA avoids many of these drawbacks and is thus an excellent basis for hierarchical self-organisation with HiSPADA. It is based on an adaptive neighbourhood relation, works with local knowledge only and is fully distributed. This distinguishes the algorithm from other approaches to partitioning where global knowledge is a prerequisite, e.g., from control theory (Motee and Sayyar-Rodsari, 2003). Apart from a fully connected acquaintances graph—which can be generated on the fly if used in conjunction with HiSPADA based on the neighbourhood—no organisational structure is required. While other approaches are often optimised to specific problems in specific domains (e.g., Younis and Fahmy, 2004) and thus might make better use of domain-specific knowledge, SPADA deals with the properties of MAS

in a generic way and allows the solution of different problems such as clustering, anti-clustering, and coalition formation by applying different metrics (i.e., reward functions). HiSPADA in turn uses these facilities to be equally versatile and applicable in a number of systems in which correct composition of a hierarchy layer, and requirements for the introduction and dissolution of hierarchy levels can be expressed as constraints. The composition constraints in turn determine which kind of set partitioning problem is solved.

Pournaras et al. (2009) propose AETOS—the Adaptive Epidemic Tree Overlay Service—as a means to create hierarchical structures in complex distributed systems that are robust to node failures. For this purpose, the tree is organised so that higher levels in the hierarchy are more robust—i.e., have a lower failure rate, corresponding to reliability (Steghöfer et al., 2010)—than the ones closer to the leafs. The technical foundation is a gossiping mechanism and a protocol not unlike that used by SPADA in which agents can send requests to change the structure and the request can either be accepted or denied depending on the qualitative change in the overall structure it would entail. Even though the paper is focused on robustness, the technical description does not prohibit using other metrics as the basis for the formation of the tree. The main differences to the work proposed here are that AETOS does not support pre-defined hierarchical structures, can not reorganise within a layer of the tree structure, can switch sub-trees from one parent to a completely different one, and that the algorithm continuously collects information about potential candidates instead of only communicating if a reorganisation is necessary.

Hierarchical self-organisation is a fundamental concept in ecology and biology. Bonabeau et al. (1996) propose a differential equation model of self-organising hierarchies in animal societies based on the interactions between animals at different levels of a hierarchy. Görnerup and Crutchfield (2008) argue that complex hierarchical structures emerge spontaneously due to self-organisation processes on simple so-called ϵ-machines. Their findings have an impact on the understanding of the origin of evolution starting in a pre-biotic age. Unfortunately, papers from these disciplines are mostly focused on explaining a single observable phenomenon rather than unearthing new paradigms with potential applications in other fields.

Neural networks are also a domain in which the term "hierarchical self-organisation" is often used, albeit in another sense as in the systems regarded here. A self-organising neural network—also called a self-organising map or a Kohonen Map after (Kohonen, 1982)—maintains the topological information of the input signals. This line of research originated in biology as well, more specifically in neurology in an effort to explain how sensory information is mapped to simplified representations by higher-order organisms. This paradigm has been used in computer science to create networks with the ability to create low-dimensional representations of complex input data. Hierarchies are introduced when self-organising maps are arranged in multiple levels (Luttrell, 1989), e.g., "to preprocess high dimensional input data into a hierarchy of reduced representations, which may then be subsequently processed in the manner of multi-resolution image processing." An advancement over these ideas is the "growing hierarchical self-organising map" (Rauber et al., 2002). This variant of the Kohonen map grows during the training process to find the ideal map size for the input data. Hierarchical relations within the data can also be represented in the structure of the map. However, a hierarchical self-organisation process in which the hierarchical structure originates from a process within the system is not present here.

Chapter Summary and Outlook

This chapter introduced two approaches to set partitioning, one solving the problem in a distributed fashion and one meta-heuristic, solving the problem with global information. Both algorithms can be re-used by the HiSPADA control loop that uses them in the self-organised formation of hierarchies in large-scale systems according to application-specific constraints. HiSPADA has been applied to the power management case study and the evaluation results show that the introduction of the hierarchy increases the scalability of the system without compromising the solution quality. Especially the latter characteristic will be discussed in depth in Chapter 10 and Chapter 11 which deal with solution approaches to hierarchical task allocation problems in systems of systems. Hierarchies are the basis for all algorithms introduced in this thesis and will thus remain in the center of attention throughout the rest of this work.

Part III

Controlled Emergence

Approaches to the specification of correct system behaviour, to observation of this behaviour at runtime, and for the generation of monitoring infrastructure from the requirements documents.

Chapter 5

Specification of Soft Constraint Problems with Constraint Relationships

Summary. This chapter introduces *constraint relationships* as a means to define qualitative preferences on the constraints of complex constraint systems. The approach is suitable for applications in dynamic and open environments with a high number of constraint in which constraint satisfaction problems (CSPs) can change at runtime. These circumstances make it very difficult for modellers to quantify preferences and impossible to maintain them manually. Constraint relationships express preferences over the satisfaction of constraints with a clear semantics without assigning priorities to concrete domain values. A CSP including a set of constraint relationships can be transformed into a k-weighted CSP as a representative of c-semirings that is solved by widely available constraint solvers. Their integration into the Restore Invariant Approach for the monitoring of correct system behaviour at runtime is illustrated.

Publication. While constraint relationships have been described in (Schiendorfer et al., 2013), the Restore Invariant Approach and its underlying formal foundations have most recently been published in (Nafz et al., 2011).

As established in Chapter 3, Systems of System can exhibit emergent behaviour. In the ideal case this behaviour fulfils the system goals and the system behaves as desired in every state of the system. However, the ever-changing environment of large-scale open heterogeneous self-organising systems and their inherent uncertainty prohibit a priori specification of possible system states. Therefore, it is not certain that the system will not show undesirable behaviour that is potentially detrimental to reaching the system goals. Therefore, the emergent behaviour of the system needs to be monitored and checked against the system goals. The infrastructure required to do this is provided by the Observer/Controller architecture implemented in the system. However, a theoretical framework needs to be in place to specify the preferential constraints that describe correct behaviour of large-scale open heterogeneous self-organising systems.

This chapter first outlines the Restore Invariant Approach (RIA), used to formally describe an adaptive system, its correct behaviour, and its reconfiguration. It then introduces constraint relationships as a means to specify soft constraint problems that allow for a fine-grained definition of correct and optimal behaviour and will be used in the following chapters both in the specification of constraints that are observed at runtime to ensure system correctness and in the specification of constraint models that allow a reconfiguration of the system in Part IV.

5.1 The Restore Invariant Approach for the Specification of Correct System Behaviour

The Restore Invariant Approach (RIA) allows defining a corridor of correct behaviour. The system tries to operate within the corridor as long as possible. Due to unexpected disturbances, the system potentially leaves the corridor. Disturbances can be changes in the environment, failures, new or leaving agents, or new objectives, for instance. Whenever the corridor is left, the system initiates a self-* phase and tries to reconfigure in order to return to the corridor. Reconfiguration of the system is further

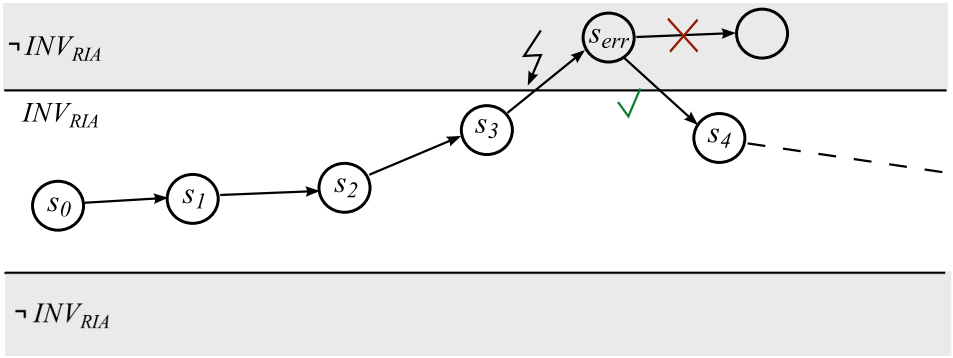


Figure 5.1: Corridor of correct behaviour of a self-* system (Nafz et al., 2011). System states either fulfil the invariant INV_{RIA} and are therefore within the corridor or violate the invariant and therefore trigger a reaction. This reaction aims to restore the system to a state in which the invariant holds again.

elaborated in Part IV. This part of the thesis is concerned with the specification of correct behaviour and the observation of the system to detect misbehaviour.

Corridors of Correct Behaviour

The basic idea behind the Restore Invariant Approach is to constrain the behaviour of the system so that it only exhibits correct behaviour. An advantage of this approach is that the system retains its flexibility and is still able to adapt during runtime and make decisions autonomously.

From a formal point of view, a system can be described as a transition system $SYS = (S, \rightarrow, I, AP, L)$, where S is the set of states, $\rightarrow \subseteq S \times S$ a transition relation, $I \subseteq S$ a set of initial states, AP a set of atomic propositions and L a labelling function. A trace π of the system is then given by a sequence of states $s_i \in S$ whose states are related by the transition relation and which starts in an initial state s_0.

$$\pi = s_0, s_1, s_2, \ldots, s_n$$

Fig. 5.1 shows an example trace of an abstract transition system SYS which tries to stay within the corridor. The system recognises a violation of the corridor and triggers a self-* process in order to reach a state within the corridor.

Formally the specification of the corridor corresponds to a predicate logic formula[1] – the invariant INV_{RIA} – which is evaluated over a system state. The term "invariant" is used as the system's goal is to maintain the invariant throughout the entire system trace.

The invariant differentiates the system states into those that exhibit correct behaviour and those that do not. This allows to separate the states into two disjoint sets: a set S_{func} of functional states within the corridor in which the system shows its desired behaviour and a set S_{reconf} of reconfiguration states outside the corridor in which reconfiguration is necessary. This abstract definition can accommodate a variety of situations in the system that can lead to adaptations. If, e.g., new agents that enter the system should trigger a reconfiguration, the invariant will have to be formulated so that an idle agent or one that has not been configured violates it. The system will then switch to a reconfiguration state as soon as such a situation occurs.

Gärtner (1999) presents a similar classification of the state space for fault tolerant systems. He distinguishes three kinds of states: a set of *invariant states*, in which the system exhibits the desired properties, corresponding to the functional states of RIA; a set of states constituting the *fault span*, containing all invariant states and additionally all states which are tolerable by the system and from which the system eventually returns into an invariant state; finally, the set of all possible states.

[1] Theoretically, a temporal logic formula could be used instead of a predicate logic formula. However, it is unclear how a system can evaluate a temporal invariant during runtime and decide whether it is violated or not. In order to decide this, the system would have to predict the future behaviour. In the area of runtime verification the correctness of temporal logic properties is checked during runtime. For example, Bauer et al. try to monitor temporal logic properties during runtime (Bauer et al., 2011). In each step the property can be true, false or inconclusive. In this chapter predicate logic is used to formulate the invariant, although the general approach is not limited to it. The use of predicate logic implies that the invariant can be evaluated in each state.

Another classification of the state space of Organic Computing systems is proposed by Schmeck et al. (2010). The *target space* contains the states the system should try to reach. If this is not possible, the system should at least try to get into a state of the *acceptance space*. The *survival space* consists of all states outside the acceptance space from which the system can get back into the acceptance or target space. All remaining states are states within the *dead space* with no possibility to get back into the acceptance space. Compared to the corridors of RIA, Schmeck et al. split the functional states into target and acceptance space to distinguish optimal and non-optimal but correct states.

Both classifications separate the reconfiguration states into a set of states in which a path back into a functional state exists and a set where no path exists any more. The classifications are used to describe the behaviour of a self-* system on an abstract level. The specification of behavioural corridors in RIA exceeds these classifications by providing the tools to clearly define the different sets of states and to use these definitions both at design time to provide techniques for formal analysis as well as at runtime to monitor the correct behaviour of the system (see Chapter 6).

Behavioural Guarantees based on RIA

By distinguishing functional and reconfiguration states, the requirements for the self-* properties of the system can be specified using the invariant. Whenever the invariant is violated, the system has to try to return to the corridor and to restore the invariant. The invariant is also a sufficient condition for system states that exhibit the expected behaviour. That means that the system exhibits correct behaviour when in a state in which the invariant holds. The correctness of the functional behaviour of the system can therefore be verified independently of the self-* mechanisms. For the verification of the functional system it is assumed that there exists a mechanism that restores the invariant when it is violated. For a specific self-* mechanism it has to be proven that this assumption holds.

The definition of corridors has several more advantages compared to an explicit listing of all states. First, it is usually hard or expensive to find and list all states that are valid. It is often easier to formulate common properties that valid states need to exhibit. The abstraction induced by the invariant reduces the complexity of formal reasoning and the separate treatment of functional properties and self-* behaviour can be exploited in order to give behavioural guarantees. For more details on these topics, please refer to (Nafz et al., 2013).

5.2 Over-constrained problems and soft constraints

The invariant is usually a complex term in conjunctive normal form, composed of a number of formulae that express different properties of the system. Each of these formulae is a single constraint that encapsulates a predicate over a system state. Not all of these formulae have the same importance. In the RIA, however, all of them bear the same significance and thus, violation of one of the constraints leads to the violation of the invariant as a whole. This view corresponds to Constraint Satisfaction Problems, sometimes also called *constraint network*, in which a valuation for variables over which hard constraints are defined is sought. We follow and extend the definitions for classical constraint networks used in (Rossi et al., 2006).

Definition 1 *A **constraint network** is represented as a triple $\langle X, D, C\rangle$: where $X = \{x_1, \ldots, x_n\}$ is a finite set of n variables; $D = \{d_1, \ldots, d_n\}$ is the set of corresponding domains such that x_i takes values from d_i; and C is a finite set of constraints. Each $c \in C$ is represented by a pair (r, v) where $v \subseteq X = sc(c)$ represents the scope of the constraint, i.e., on which variables the constraint is defined, and $r \subseteq \prod_{x_i \in v} d_i = rl(c)$ is a relation between the variables contained in v that defines the allowed combinations for c.*

Assignments *are tuples $t_V \in \prod_{x_i \in V} D_i$ with $V \subseteq X$. If $W \subseteq V$, $t_V[W]$ returns the* projection *of a tuple containing only elements in W. An assignment is* complete *if $V = X$, and* partial *otherwise. A constraint $c \in C$ is* fully assigned *by t_V if $sc(c) \subseteq V$. An assignment is* consistent with *c if c is fully assigned by t_V and $t_V[sc(c)] \in rl(c)$; we then write $t_V \models c$. An assignment is a* solution *if $\forall c \in C : t_V \models c$. We often omit the variables from assignments and write t instead of t_V.*

In many systems, however, such a rigid formulation is not flexible enough. Sometimes, it is desirable to solve a constraint satisfaction problem as good as possible (i.e., accepting that some constraints might be violated, cf. Section 9.2) or—when referring to the corridors of correct behaviour introduced

above—to only react to the violation of several constraints at once or to only reconfigure if a constraint has been violated for a certain period of time (cf. Section 6.2). This is due to the fact that many constraint problems are *over-constrained*, i.e., there is no solution that fulfils all constraints. Especially if some constraints do not express physical limitations but desirable assignments, a more graduated approach is required.

In such cases, not all specified constraints will be part of the invariant. Instead, a core of constraints that will always have to hold—the *hard* constraints—will form the invariant while other, less important constraints that do not warrant an immediate reconfiguration in case of their violation, compose a separate, *soft* constraint problem.

Definition 2 *A **soft constraint network** is represented as a triple $\langle X, D, C\rangle$ as above but with $C_h, C_s \subseteq C$ the sets of hard and soft constraints, respectively, such that $C_h \cup C_s = C$ and $C_h \cap C_s = \emptyset$. An assignment is then a* solution *if $\forall c \in C_h : t_V \models c$, i.e., all* hard *constraints are fulfilled.*

The simplest form of a soft constraint problem is a weighted constraint network, in which each constraint is assigned a weight that can be interpreted as a penalty for the violation of the constraint. In order to denote hard constraints in a similar way, a constant k is introduced as an upper bound of the penalty.

Definition 3 *A* weighted constraint network *$\langle X, D, C, w\rangle$ is given by a constraint network $\langle X, D, C\rangle$ and a* weighting function *$w : C \to \mathbb{R}_{\geq 0}$. The weight of an assignment t_V is the combined weight of all unsatisfied constraints. The purpose of a weighted CSP is to find assignments having minimal weight. A violation greater than k is considered unacceptable and $\forall c \in C_h : w(c) = k$.*

$$w(t_V, C' \subseteq C) := \sum_{c \in C' : t_V \not\models c} w(c)$$
$$w(t_V) := w(t_V, C)$$

By this definition, it follows that:

$$\forall C_1, C_2 \subseteq C : C_1 \uplus C_2 = C \to w(t_V, C_1) + w(t_V, C_2) = w(t_V, C) \tag{5.1}$$

We write $C_1 \uplus C_2$ to denote the *disjoint union* of C_1 and C_2, where we require that $C_1 \cap C_2 = \emptyset$.

Overview of Soft Constraint Formalisms

Several formalisms to express soft constraints and thus preferences over constraints or solutions have been devised (Rossi et al., 2006). Among these are several kinds of soft constraints such as Fuzzy CSP (Dubois et al., 1993; Gelain et al., 2007) or Weighted CSP (WCSP) (Shapiro and Haralick, 1981) as well as other mechanisms like conditional preference networks (CP-nets) (Boutilier et al., 2004) or constraint hierarchies (Borning et al., 1992). Many of these are discussed in more detail in Section 5.5, while the following gives a short overview of the different approaches.

Generic frameworks for soft constraints such as c-semirings (Bistarelli et al., 1995) or valued constraints (Schiex et al., 1995) have been proposed to design generic algorithms and prove useful properties over a common structure. Assignments are labelled with preference levels or violation degrees that are combined using a specific operator to find preferred solutions. Formally, a *c-semiring* $\langle E, +_s, \times_s, \mathbf{0}, \mathbf{1}\rangle$ comprises a set of preference levels E; a binary operation $+_s$ closed in E used to compare assignments (defining $e \geq_s e' \leftrightarrow e +_s e' = e$) for which $\mathbf{0}$ is a neutral element (i.e., $e +_s \mathbf{0} = e$) and $\mathbf{1}$ is an annihilator (i.e., $e +_s \mathbf{1} = \mathbf{1}$); and a binary operation $\times_s$ also closed in E used to combine preference levels where $\mathbf{0}$ is an annihilator and $\mathbf{1}$ a neutral element.

In WCSPs, a cost function is provided for each constraint. The function assigns a weight from $\mathbb{R}_{\geq 0}$ to each tuple of values, corresponding to the definition of a weighted constraint network above. Solving a WCSP then consists of minimizing the sum of weights of all violated constraints. This basic form has been refined to *k-weighted CSPs* that include a special weight k for hard constraints, meaning that assignments over k are unacceptable (Larrosa, 2002). The weight k takes the role of the maximal element with respect to $\geq_s$ in the corresponding c-semiring which is defined as $\langle 0..k, \min, +^k, k, 0\rangle$ (with

$0..k = \{0, \ldots, k\}$ and $x +^k y = \min\{k, x + y\}$). This semiring instance will also be used to implement the preference semantics of constraint relationships for uses in standard constraint solvers as described in Section 5.4.

In *Fuzzy CSPs* (Dubois et al., 1993; Gelain et al., 2007), the satisfaction degree of an assignment with respect to a constraint is expressed according to the theory of fuzzy sets. The individual membership functions of the constraints (representing how strongly an assignment is an element of the constraint's fuzzy set) are conjoined by taking the minimum of their satisfaction degrees.

In *constraint hierarchies* (Borning et al., 1992), constraints are categorised into strict levels, represented as sets $H_0, \ldots, H_n$ in which constraints in H_i take precedence over constraints in H_{i+1} (i.e., level i dominates level $i + 1$). Valid solutions fulfil all constraints in H_0 and as many as possible in the dominated levels. Constraint hierarchies offer a similar design paradigm to constraint relationships. This formalism is suited well for problems that incorporate a "totalitarian" semantics (Brafman and Domshlak, 2009)—i.e., it is better to satisfy a very important constraint rather than many less important ones. We argue that different semantics are required for other, more "egalitarian" problem classes. In constraint relationships, indifference is expressed by leaving out orderings—hierarchies only enable this for constraints on the same level. Furthermore, we show in Section 5.5 that constraint relationships can express a particular class of constraint hierarchies, namely *locally-predicate-better* hierarchies, but model additional solution preferences.

CP-nets (Boutilier et al., 1997) are a qualitative tool to represent preferences over assignments by specifying orders over domain values. They intend to reduce the complexity of all possible preference orderings by structuring the variables according to their mutual impact. Concretely, it is assumed, that only some variable assignments affect the preference order on the domain values of others. Given this structural information about influences, a decision maker is asked to explicitly specify her preferences over the values of a variable X for *each* assignment to its *parent* variables, i.e., the ones that affect the preference on the domain of X. Hence, a set of total orders on the values of finite domains is kept for every variable in a conditional preference table. Lifting these orders to solutions generally results in a preorder (Rossi et al., 2008). While many real-world examples can be adequately modelled using CP-nets if user preferences on the actual values can be elicited (Boutilier et al., 1997), we argue that for complex constraint networks with infinite domains it is easier to make generalizing statements (Brafman and Domshlak, 2009) that refer to a coarser level of granularity—preferences on constraints rather than on domain values for single variables. As the number of constraints is arguably lower than the number of variables and possible domain values for larger problems, the definition of preferences over constraints simplifies the preference elicitation problem considerably. Whereas CP-nets offer efficient solving algorithms due to the extensional structure of variables and values, our approach works with intensional constraints as well. A comparison to CP-nets can be found in Section 5.5.

An Illustrative Example of Soft Constraints with Constraint Hierarchies

To illustrate the problems associated with current approaches to soft constraints, we will use the "What to Wear" example, described in (Freuder and Wallace, 1992), for instance. It describes appropriate fashion choices for a male protagonist, originally an autonomous robot, albeit the constraints used can also be of help for a member of the human species. The robot has to choose proper attire for the day and is faced with a selection of shirts, footwear and trousers. These selections can be expressed as finite domains $S = \{r, w\}$ for a red or white shirt, $F = \{l, s\}$ for leather shoes or sneakers, and finally $T = \{b, d, g\}$ for blue trousers, denims, or grey dress pants.

The fashion-conscious robot is equipped with a number of constraints indicating sensible combinations of the garments available. These can be expressed by giving pairs of garments that work well together. For example, a constraint that tells the robot that a red shirt will work well with grey dress pants can be expressed as[2] $c_1 : (r, g)$. A complete set of constraints could look like this:

[2]The constraints used in this example are expressed as binary extensional constraints, i.e. two-valued tuples over values of the variables. The proposed techniques are independent of the way constraints are formulated. Non-binary CSPs can be converted into equivalent binary CSPs (Bacchus and van Beek, 1998). The constraint $c_1 : (r, g)$ states that a red shirt and grey pants work together. The semantics of this statement are inconsistent in the literature. Here, we use the following interpretation: the choices are regarded as predicates and equivalences are used: $c_1^p : r \leftrightarrow g$. If several constraints are defined, all constraints have to hold unless otherwise stated (And-semantics). The latter semantics differs from the one used, e.g., in (Borning et al., 1992), but is concise and intuitive in the context presented here.

$$c_1 : (r, g) \quad c_2 : (w, b) \quad c_3 : (w, d)$$
$$c_4 : (s, d) \quad c_5 : (l, g) \quad c_6 : (w, l)$$

Interpreting all above constraints as hard ones, i.e., constraints that have to be fulfilled under any circumstances will not yield a solution. If a white shirt is chosen, $c_2 : (w, b)$ and $c_3 : (w, d)$ exclude each other. If a red shirt is chosen, $c_1 : (r, g)$ forces grey pants. These in turn require leather shoes due to $c_5 : (l, g)$ which require a white shirt according to $c_6 : (w, l)$. The problem is thus over-constrained. In classical PCSP approaches, the solver would interpret all constraints as soft and thus try to find a solution to a relaxed problem consisting only of a subset of the original constraints (Freuder and Wallace, 1992). A more directed course of action can be achieved by defining which constraints are hard, i.e., may not be violated and are thus elements of the set of hard constraints C_H, and which ones are soft, i.e., may be violated if absolutely necessary and are elements of the set of soft constraints C_S. This can be done by introducing constraint hierarchies (Borning et al., 1992) that categorize the constraints into appropriate levels in the hierarchy. The "required" level indicates that all constraints in it are hard and may not be violated under any circumstances. "Strong" constraints need to be observed as long as possible and "weak" constraints are the first that are dropped while looking for a solution. Due to these semantics, the above set of constraints has to be slightly altered: as it contains two choices of pants for a white shirt, c_2 and c_3 need to be combined. The problem can then be formulated as follows[3]:

$$\begin{aligned} \text{required}: &\quad c_1 : (r, g) \quad c_2^* : (w, \{b, d\}) \\ \text{strong}: &\quad c_4 : (s, d) \quad c_5 : (l, g) \\ \text{weak}: &\quad c_6 : (w, l) \end{aligned}$$

Now, the problem becomes solvable, as the conflict in the hard constraints has been resolved and soft constraints can be omitted. This ability, however, comes at the expense of sacrificing the weak constraint that proposes to combine the white shirt with leather shoes. The robot's constraint solver will find that both the combination of a red shirt, leather shoes, and grey dress pants (r, l, g), as well as the combination of a white shirt, sneakers, and denim trousers (w, s, d) are appropriate.

The decision whether a constraint is hard or soft is the result of a process called "preference elicitation" (see, e.g., Gelain et al., 2007). After it has been detected that a problem is over-constrained, the modeller or the user of the system has to provide information to distinguish the constraints and to establish preferences on them. In the case of constraint hierarchies, as above, the result of the process is a hierarchical relationship between the constraints. In other cases, such as fuzzy or weighted constraints, the modeller or user has to provide a level of preference for specific assignments (for Fuzzy CSPs) or weights for each constraint (for Weighted CSPs). The order of these values is equivalent to the preference over the constraints. Such an approach makes large problems very difficult to handle. Assigning numbers to each constraint might introduce implicit preferences the modeller is not aware of and might not want. It is also difficult to extend the problem at a later point in time, especially if dealing with flexible problems in which constraints can be added and removed or changed while being in active use in a running system. The consistency of the constraint system might thus be jeopardized each time the system changes and it becomes burdensome to maintain such a system.

Example—Ski-Day Planner
As an additional example (taken from Schiendorfer et al., 2013), consider an application that guides travellers exploring a new ski area by offering a plan for a ski day. Each skier has different priorities that can be set interactively. Assume the following soft constraints are defined on the set of possible tours (that need to respect hard constraints such as weather induced blockages or daylight time):

- Avoid black slopes (ABS): Beginners avoid difficult (marked "black") slopes.
- Variety (VT): Different slopes should be explored.
- Fun-park (FP): A feature for Freestyle fans

[3]The formulation used here is akin to the one from the "Guide to Constraint Programming" by Roman Bartak, available online at `http://kti.mff.cuni.cz/~bartak/constraints/`. It has been adapted to fit the equivalence semantics used here.

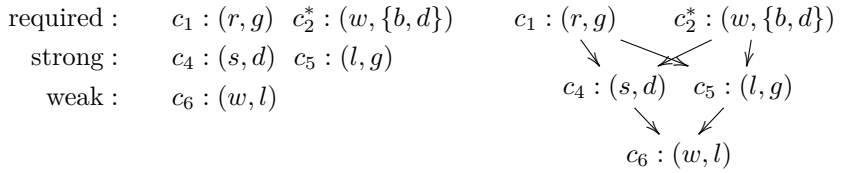


Figure 5.2: The constraint hierarchy for "What to Wear" and the corresponding preference graph.

- Little Wait (LW): Impatient visitors prefer not to wait too long at a lift.
- Only Easy Slopes (OE): People restrict their tours to easy ("blue") slopes.
- Lunch Included (LI): Some travellers enjoy a good mountain dish.

For clarity, we abstract from details such as concrete tuple representations and leave hard constraints aside by assuming the following three assignments are solutions but differ in their performance on soft constraints.

- $t_X^{(1)} \models \{\mathsf{LW}, \mathsf{OE}, \mathsf{LI}, \mathsf{FP}\} \wedge t_X^{(1)} \not\models \{\mathsf{VT}, \mathsf{ABS}\}$
- $t_X^{(2)} \models \{\mathsf{VT}\} \wedge t_X^{(2)} \not\models \{\mathsf{FP}, \mathsf{ABS}, \mathsf{LW}, \mathsf{LI}, \mathsf{OE}\}$
- $t_X^{(3)} \models \{\mathsf{OE}, \mathsf{FP}, \mathsf{ABS}, \mathsf{LI}\} \wedge t_X^{(3)} \not\models \{\mathsf{VT}, \mathsf{LW}\}$

5.3 Expressing Preferences over Constraints with Constraint Relationships

The hierarchy for the "What to wear" example proposed above has an interesting property: it introduces a relation on the constraints. It is possible to give a "is more important than" relation for each of the constraints with each other constraint on the level(s) above or below it. Constraint $c_1 : (r, g)$ is thus more important than $c_4 : (s, d)$ and $c_5 : (l, g)$. The former two are, on the other hand, more important than $c_6 : (w, l)$. As the relation may be transitive, c_1 is also more important than c_6. Pairs of constraints on the same level of the hierarchy are equally important. This relation can be expressed as a preference graph where the direction of the edges reflects the relation. For the example, such a graph is depicted in Fig. 5.2.

An alternative way to express the preference relation is by explicitly providing a *preferential constraint relationship*. This can be expressed by the $\succ_R$ operator on constraints that signifies that one constraint "is more important than" another. For the example of the fashion-conscious robot, the preferences can be expressed as:

$$\begin{array}{lll} c_1 \succ_R c_4 & c_2^* \succ_R c_4 & c_4 \succ_R c_6 \\ c_1 \succ_R c_5 & c_2^* \succ_R c_5 & c_5 \succ_R c_6 \end{array}$$

As the order is partial, it is possible to leave the specification ambiguous in cases where it is either impractical or undesired to compare two constraints, as, e.g., for c_4 and c_5.

Constraint relationships constitute a qualitative "is more important than"-relation over constraints to denote which constraints should rather be satisfied and which ones can be dropped or disregarded if necessary. We argue that this is a useful and realistic approach that enables an easy-to-use formalism for special application areas and positions itself among the AI formalisms introduced earlier and discussed in Section 5.5. Constraint relationships are targeted at over-constrained problems and problems with many constraints where preference levels are hard to maintain manually, or dynamic constraint satisfaction problems with frequently changing constraints and preferences. Typical applications facing these circumstances are multi-agent systems with incomplete preferences (Rossi, 2008). The combination of different constraint models in a synthesis process in hierarchical or holonic systems as described in Chapter 10 is another application area of the approach.

The corridors of correct behaviour also benefit from such a formalism: since it becomes possible to specify the importance of constraints, different degrees of optimality within the corridor can be distinguished. A system state can thus be evaluated not only for its validity but also for its quality.

Hard constraints can be defined on the maximum weight of the soft constraints that can be violated at the same time. These facilities are exploited in the observation of the system and in self-optimisation as detailed in Section 6.2.

To be able to formally define constraint relationships, some properties of binary relations need to be defined first.

Definition 4 *A binary relation $Q \subseteq M \times M$ on a set M is* asymmetric *if $(m, m') \in Q$ implies that $(m', m) \notin Q$; it is* transitive *if $(m, m') \in Q$ and $(m', m'') \in Q$ implies $(m, m'') \in Q$; it is a* partial order relation *if it is asymmetric and transitive. The* transitive closure *of Q, denoted by Q^+, is inductively defined by the rules that (a) if $(m, m') \in Q$, then $(m, m') \in Q^+$ and (b) if $(m, m') \in Q$ and $(m', m'') \in Q^+$, then $(m, m'') \in Q^+$.*

We can now define constraint relationships on soft constraints based on these terms.

Definition 5 *A set of* constraint relationships *for the soft constraints C_s of a constraint network $\langle X, D, C\rangle$ is given by a binary asymmetric relation $R \subseteq C_s \times C_s$ whose transitive closure R^+ has to be a partial order relation. We write $c \succ_R c'$ or $c' \prec_R c$ iff $(c, c') \in R$ to define c to be* more important *than c', analogously for R^+. If $c' \prec_R c$ we call c' a* direct predecessor*, if $c' \prec_{R^+} c$ a* transitive predecessor *of c. Moreover, we refer to the* constraint relationship graph *as the directed graph spanned by $\langle C_s, R\rangle$.*

Extending the Example

The benefits of expressing the relation in the way described above becomes clear if the simple "What to Wear" example is extended with a new domain and additional constraints. Consider an additional garment, a jacket. The robot has a choice of a black jacket and a corduroy one, giving an additional domain $J = \{bl, co\}$. The new choice also comes with a number of constraints:

- It is appropriate to wear the red shirt with the black jacket $c_7 : (r, bl)$.
- The white shirt goes well with the corduroy jacket $c_9 : (w, co)$ or the white shirt $c_8 : (w, bl)$.
- Fortunately, both jackets go with all the shoes $c_{10} : (bl, s)$, $c_{11} : (bl, l)$, $c_{12} : (co, l)$, $c_{13} : (co, s)$.

To maintain the correct equivalence semantics, c_8 and c_9 are combined to $c_8^* : (w, \{bl, co\})$. The same combination is applied to c_{10} and c_{11} as well as c_{12} and c_{13} to yield $c_{10}^* : (bl, \{s, l\})$ and $c_{12}^* : (co, \{s, l\})$.

Unfortunately, these additional constraints only serve to make the problem more complicated. The important question for the robot is thus on which level in the hierarchy the new constraints should be integrated. One way to do this is to annotate each constraint with the level it belongs to. In order to this, preference elicitation is required and the modeller would have to make choices for each constraint. Every choice influences the set of possible outcomes, and might even make the problem unsolvable in case a contradiction is introduced at the required level. The best chance for the modeller would be to iteratively add constraints on their appropriate levels and check the system to see whether there still are acceptable solutions. Clearly, this is not an efficient and productive way to go about solving the problem. It also requires global knowledge, as all constraints and their respective levels have to be known for an informed choice.

If, however, the modeller uses constraint relationships, the problem becomes much easier to handle. Instead of classifying the constraint into the hierarchy, the modeller just has to give preferences with regard to other constraints, thus reducing the problem from one that requires a global view to one where only some of the constraints need to be known. The preferences can be gained easily from a magazine dealing with menswear. A quick look at the literature (Esquire Magazine, 2009) reveals the following preferences:

1. Sneakers work better with a black jacket than with a corduroy jacket:
   $c_{10}^* : (bl, \{s, l\}) \succ_R c_{12}^* : (co, \{s, l\})$
2. It is preferable to wear the black jacket with the red shirt over wearing any jacket with the white shirt: $c_7 : (r, bl) \succ_R c_8^* : (w, \{bl, co\})$
3. A well-dressed men would sacrifice wearing grey pants with a red shirt if he could wear a black jacket with a red shirt: $c_7 : (r, bl) \succ_R c_1 : (r, g)$

$$
\begin{aligned}
c_1 : (r, g) &\succ_R c_4 : (s, d) & \qquad c_5 : (l, g) &\succ_R c_6 : (w, l) \\
c_1 : (r, g) &\succ_R c_5 : (l, g) & \qquad c_7 : (r, bl) &\succ_R c_1 : (r, g) \\
c_2^* : (w, \{b, d\}) &\succ_R c_4 : (s, d) & \qquad c_7 : (r, bl) &\succ_R c_8^* : (w, \{bl, co\}) \\
c_2^* : (w, \{b, d\}) &\succ_R c_5 : (l, g) & \qquad c_{10}^* : (bl, \{s, l\}) &\succ_R c_4 : (s, d) \\
c_4 : (s, d) &\succ_R c_6 : (w, l) & \qquad c_{10}^* : (bl, \{s, l\}) &\succ_R c_{12}^* : (co, \{l, s\})
\end{aligned}
$$

Figure 5.3: The extended system of constraint relationships for "What to Wear" with jackets.

4. Another sacrifice the connoisseur is willing to make is to wear denims with his sneakers if it means he can wear his black jacket: $c_{10}^* : (bl, \{s, l\}) \succ_R c_4 : (s, d)$

These preferences, depicted in the context of the existing ones in Fig. 5.3, can no longer be expressed within the bounds of the existing hierarchy. Especially the relationship $c_7 : (r, bl) \succ_R c_1 : (r, g)$ makes this impossible as it would require the introduction of an additional level in the hierarchy. Also note that there are now constraints that are not in a relationship with any other (such as c_{11}). It is not clear on which level these should be made part of a hierarchy.

In order to make the relationships usable for a solver, they now have to be transformed into weights that respect a partial order over the constraints. This order can be expressed as a directed, acyclic graph, allowing to check the system for consistency. If the order does not constitute a DAG, the modeller can be alerted to cyclic relationships (see Section 5.4). The solvable model is created by using the DAG to annotate each constraint with a weight, indicating the penalty for an unsatisfied constraint (Section 5.4). This way, the partial order that has been established by the relationships is automatically transformed into a total order as it is used in Weighted CSPs.

An application of constraint relationships to the scheduling problem in the autonomous power management case study is illustrated in Section 9.2 where constraint relationships are used to define preferred operation states of power plants and limit, e.g., the rate of change to increase efficiency.

Example—Ski-day planner
Assume three personas as prototypical customers: Skier A is impatient, skilled in skiing, wants to explore a fun-park but is not afraid of difficult slopes or needs lunch since he wants his workout. Boarder B is an explorer, she wants a large number of different slopes (except for black ones) but accepts to wait. Rookie C started skiing and wants to avoid black slopes. He appreciates a tour of easy slopes and lunch. These preferences can easily be encoded in constraint relationships:

$$
\begin{aligned}
\text{Skier}A: \quad & LW \succ_R FP \quad LW \succ_R ABS \quad LW \succ_R OE \quad LW \succ_R FP \\
& FP \succ_R LI \quad VT \succ_R LI \quad OE \succ_R LI \\
\text{Boarder}B: \quad & VT \succ_R FP \quad VT \succ_R ABS \quad FP \succ_R LI \quad ABS \succ_R LW \\
& LI \succ_R LW \quad LW \succ_R OE \\
\text{Rookie}C: \quad & ABS \succ_R LW \quad ABS \succ_R LI \quad ABS \succ_R OE \quad LW \succ_R VT \\
& LI \succ_R FP \quad OE \succ_R LI \quad OE \succ_R VT \quad VT \succ_R FP
\end{aligned}
$$

Semantics of dominance properties

We have defined constraint relationships syntactically as a relation that denotes some constraint being "more important" than another one. However, we need to express *how much* more important a constraint is than another one in order to address questions such as "Is it better to satisfy a more important constraint than *all* its less important predecessors?". Concretely, we examine ways to lift the binary relation over soft constraints to sets of soft constraints that are violated by an assignment. Such a *violation set* is denoted by capitalizing the letter used for the assignment; i.e., for some assignment t its violation set is $T = \{c \in C_s \mid t \not\models c\}$.

We consider several possibilities for worsening a violation set and will use $T \longrightarrow_R^p U$ to express that T is worsened to U by using the strategy or *dominance property* p. All strategies share two generic rules that we present first. On the one hand, a set of violated constraints T gets worse if some additional

constraint c is violated:

$$T \longrightarrow_R^p T \uplus \{c\} \tag{W1}$$

On the other hand, worsening two independent parts of a violation set leads to a worsening of the whole violation set: If T_1 is worsened to U_1 and T_2 is worsened to U_2, then $T_1 \uplus T_2$ is also worsened to $U_1 \uplus U_2$:

$$\frac{T_1 \longrightarrow_R^p U_1 \quad T_2 \longrightarrow_R^p U_2}{T_1 \uplus T_2 \longrightarrow_R^p U_1 \uplus U_2} \tag{W2}$$

In fact, we can derive

$$T \longrightarrow_R^p T \uplus \{c_1, \ldots, c_k\} \tag{W1'}$$

We have $T \longrightarrow_R^p T \uplus \{c_1\}$ by (W1) and $\emptyset \longrightarrow_R^p \{c_2\}$ again by (W1), and thus $T \longrightarrow_R^p T \uplus \{c_1, c_2\}$ by (W2), from which the result follows by induction.

We now introduce three particular dominance properties. In the first approach, violating a less important constraint rather than an important one should be considered *better—ceteris paribus*. We call this criterion *single predecessor dominance* (SPD):

$$T \uplus \{c\} \longrightarrow_R^{\text{SPD}} T \uplus \{c'\} \quad \text{if } c \prec_R c' \tag{SPD}$$

For instance, if $C_s = \{\mathsf{a}, \mathsf{b}\}$ and $\mathsf{a} \succ_R \mathsf{b}$ (constraint a is considered more valuable than constraint b), $\{\mathsf{a}\} \longrightarrow_R^{\text{SPD}} \{\mathsf{a}, \mathsf{b}\}$ by (W1) (with $p = \text{SPD}$), $\{\mathsf{b}\} \longrightarrow_R^{\text{SPD}} \{\mathsf{a}\}$ by (SPD). Since SPD does not have a single constraint dominate a set of others it is well suited for "egalitarian" problems. It is therefore similar to a MaxCSP instance where we are interested in satisfying a large number of constraints rather than discriminating strongly by their individual importance.

However, a stronger notion is needed when some constraints contribute more to the quality of a solution than a whole set of others—in particular the constraints that are explicitly denoted less important. This property is called *direct predecessors dominance* (DPD) and is motivated by the fact that human preference decisions can be intransitive (Andréka et al., 2002):

$$T \uplus \{c_1, \ldots, c_k\} \longrightarrow_R^{\text{DPD}} T \uplus \{c'\} \quad \text{if } \forall c \in \{c_1, \ldots, c_k\} : c \prec_R c' \tag{DPD}$$

For a minimal example, consider $C_s = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$ and $\mathsf{a} \succ_R \mathsf{b}$, $\mathsf{a} \succ_R \mathsf{c}$. Then violating a is more detrimental to a solution than any combination of b and c, e.g., $\{\mathsf{b}, \mathsf{c}\} \longrightarrow_R^{\text{DPD}} \{\mathsf{a}\}$. It follows by definition that $T \longrightarrow_R^{\text{SPD}} U$ implies $T \longrightarrow_R^{\text{DPD}} U$.

The most "hierarchical" notion, *transitive predecessors dominance*, consists of extending DPD to include transitive predecessors as well. It is motivated by the natural extension of constraint relationships R to its transitive closure R^+ to obtain a partial order and the ability to express a subset of constraint hierarchies with constraint relationships. Note that this property could also be achieved by using DPD rules and R^+ explicitly in the model:

$$T \uplus \{c_1, \ldots, c_k\} \longrightarrow_R^{\text{TPD}} T \uplus \{c'\} \quad \text{if } \forall c \in \{c_1, \ldots, c_k\} : c \prec_{R^+} c' \tag{TPD}$$

If $C_s = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$ and $\mathsf{a} \succ_R \mathsf{b}$, $\mathsf{b} \succ_R \mathsf{c}$, then $\{\mathsf{b}, \mathsf{c}\} \longrightarrow_R^{\text{TPD}} \{\mathsf{a}\}$, but also $\{\mathsf{c}\} \longrightarrow_R^{\text{TPD}} \{\mathsf{b}\}$. Again, by definition $T \longrightarrow_R^{\text{DPD}} U$ implies $T \longrightarrow_R^{\text{TPD}} U$. As mentioned before, $T \longrightarrow_R^{\text{TPD}} U$ iff $T \longrightarrow_{R^+}^{\text{DPD}} U$.

Each relation $\longrightarrow_R^p$ over assignments induced by the used semantics describes *how* an assignment is worsened. It only is a partial order if the generating relation (R or R^+) already is a partial order. In general, the relation does not need to be transitive as this property is not required for R. Consider for example $C_s = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$, $\mathsf{a} \succ_R \mathsf{b}$, $\mathsf{b} \succ_R \mathsf{c}$, and an SPD semantics; then $\{\mathsf{c}\} \longrightarrow_R^{\text{SPD}} \{\mathsf{b}\}$ and $\{\mathsf{b}\} \longrightarrow_R^{\text{SPD}} \{\mathsf{a}\}$, but not $\{\mathsf{c}\} \longrightarrow_R^{\text{SPD}} \{\mathsf{a}\}$ since $\mathsf{a} \not\succ_R \mathsf{c}$.

We can enforce partial orders on assignments for each dominance property $p \in \{\text{SPD}, \text{DPD}, \text{TPD}\}$, denoted by $t >_R^p u$ and to be read as "t is better than u", using $T\ (\longrightarrow_R^p)^+\ U$ (meaning repeated sequential application of the rules); we will prove the asymmetry of these transitive closures in the next section using weights.

5.4 Transforming Problems with Constraint Relationships to k-Weighted CSPs

Once the constraint problem and its constraint relationships are well-defined and the dominance property is chosen, this information can be used to transform the problem into a k-weighted CSP that can be solved with standard solvers or special purpose weighted CSP-algorithms. The transformed problem can also be used to define soft behavioural corridors as outlined in Chapter 6.

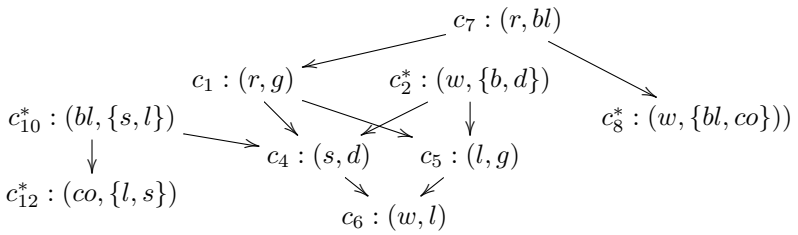


Figure 5.4: The directed acyclic graph representing the constraint relationship system depicted in Figure 5.3

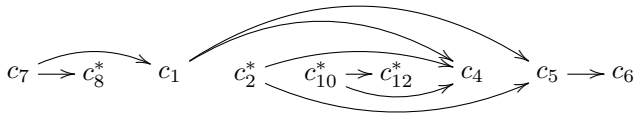


Figure 5.5: The topological sorting of the DAG in Fig. 5.4.

Checking the Relationships for Consistency

In Fig. 5.2, the preferences of the original "What to Wear" problem were expressed as a directed acyclic graph (DAG) in which each node in the graph represents a constraint while each edge represents a relationship with the direction of the edges representing the preference. Indeed, constraint relationships induce a partial order on the constraints that can be depicted as a DAG if the constraint system is consistent. The resulting graph is not necessarily weakly connected but it should be free of cycles. A cycle in the graph would indicate an inconsistent specification in which constraints are related to each other in a way that would violate the order. Formally, R^+ would then no longer be asymmetric. As it is easy to check a graph for cycles, e.g., by sorting the graph topologically (Tarjan, 1976) in $O(|V| + |E|)$, consistent specifications can be assured. The DAG corresponding to the constraint relationship system of Figure 5.3 and a topological sorting of the system are depicted in Figure 5.4 and Figure 5.5, respectively. It becomes clear from the graph that there no longer are clear separation lines that could be used to distinguish different levels of a hierarchy.

Weighting the Constraints

Algorithm 1 calculates a set of weight assignments $w \in \mathbb{N}$ for a set of constraint relationships R, and dominance property $p \in \{\mathrm{SPD}, \mathrm{DPD}, \mathrm{TPD}\}$. For each of these dominance properties we devise a weighting function $w_R^p : C \to \mathbb{N} \setminus \{0\}$ below which we prove to be strictly monotonic w.r.t. to the dominance property p for soft constraints in the sense that $\sum_{c \in T} w_R^p(c) < \sum_{c \in U} w_R^p(c)$ if $t >_R^p u$. Given the selected dominance property and the weighting function, the weight is calculated so that the dominance property holds. The algorithm terminates, as C and hence R are finite sets and R^+ is acyclic. For each hard constraint $c \in C_h$, we set $w_R^p(c) = k_R^p$ with $k_R^p = 1 + \sum_{c \in C_s} w_R^p(c)$; therefore a solution of $\langle X, D, C \rangle$ satisfying only hard constraints still has a combined weight less than k_R^p. In particular, we thus can use the c-semiring $\left\langle 0..k_R^p, \min, +^{k_R^p}, k_R^p, 0 \right\rangle$ for solving the soft-constraint problem, for which solvers are readily available.

Weighting functions

It has to be noted that by using weights, all solutions become comparable due to the totality of the order on the accumulated weights, though different solutions may show the same accumulated weight. This "defect" is not an inherent property of constraint relationships and their induced orderings on solutions, but is an artefact of the transformation into k-weighted CSPs. It would be interesting to find a c-semiring for representing constraint relationships that does not introduce such artificial orderings.

Each w_R^p will be defined recursively relying on the weights of predecessors w.r.t. R to have been already calculated. This is well defined since C and hence R are finite and R^+ is a partial order (i.e., the graph of R^+ is acyclic). Algorithmically, the weights can be determined in a bottom-up fashion or using a depth-first strategy as shown in Algorithm 1.

Algorithm 1 Dominance preserving weight assignment based on constraint relationships

```
1: w ← ∅
2: for all {c ∈ C_s : ∃c' : c ≻_R c' ∧ ∄c'' : c'' ≻_R c} do
3:     Assign-Weight(c, C, R, w)
4: end for
5: k ← 1 + Σ_{c∈C_s} w(c)
6: for all {c ∈ C_h} do
7:     w ← w ∪ (c, k)
8: end for
9: return ⟨w, k⟩
10:
11: procedure Assign-Weight(c, C, R, w)
12:     if ∄n : (c, n) ∈ w then
13:         for all {c' ∈ C : c ≻_R c'} do
14:             Assign-Weight(c', C, R, w)
15:         end for
16:         w_c ← w^p_R(c)
17:         w ← w ∪ (c, w_c)
18:     end if
19: end procedure
```

For establishing the respective strict monotonicity properties, let $W^p_R(T) = \sum_{c\in T} w^p_R(c)$; we then have to prove $W^p_R(T) < W^p_R(U)$ if $t >^p_R u$. Since $>^p_R$ is defined via $(\longrightarrow^p_R)^+$ and all dominance properties are defined inductively by rules, it suffices to show $W^p_R(T) < W^p_R(U)$ for each rule application $T \longrightarrow^p_R U$.

Rule (W1) says that $T \longrightarrow^p_R T \uplus \{c\}$; and indeed, $W^p_R(T) < W^p_R(T \uplus \{c\})$, since all weights are in $\mathbb{N} \setminus \{0\}$. Rule (W2) has the premises $T_i \longrightarrow^p_R U_i$ for $i \in \{1, 2\}$; these amount to the assumptions $W^p_R(T_i) < W^p_R(U_i)$, from which we can conclude that $W^p_R(T_1 \uplus T_2) = W^p_R(T_1) + W^p_R(T_2) < W^p_R(U_1) + W^p_R(U_2) = W^p_R(U_1 \uplus U_2)$. It thus remains to prove the strict monotonicity of (SPD), (DPD), and (TPD).

Single predecessor dominance. For SPD, we propose the function that takes the maximum weight of its predecessors and adds 1 (we take $\max(\emptyset)$ to be 0):

$$w^{\mathrm{SPD}}_R(c) = 1 + \max\{w^{\mathrm{SPD}}_R(c') \mid c' \in C_s : c \succ_R c'\} \quad \text{for } c \in C_s.$$

This mapping is indeed strictly monotonic for applications of rule (SPD): Let $T \uplus \{c\} \longrightarrow^{\mathrm{SPD}}_R T \uplus \{c'\}$ with $c \prec_R c'$. Then $w^{\mathrm{SPD}}_R(c) < w^{\mathrm{SPD}}_R(c')$ and hence $W^{\mathrm{SPD}}_R(T \uplus \{c\}) = W^{\mathrm{SPD}}_R(T) + w^{\mathrm{SPD}}_R(c) < W^{\mathrm{SPD}}_R(T) + w^{\mathrm{SPD}}_R(c') = W^{\mathrm{SPD}}_R(T \uplus \{c'\})$.

Direct predecessors dominance. For DPD we take the sum of weights of all predecessors and add 1 (summation over an empty index set is taken to be 0):

$$w^{\mathrm{DPD}}_R(c) = 1 + \textstyle\sum_{c' \in C_s : c \succ_R c'} w^{\mathrm{DPD}}_R(c') \quad \text{for } c \in C_s.$$

Rule (DPD) requires that violating a single constraint is worse than violating all its direct predecessors and hence this weight assignment assures that the weight of a constraint is strictly greater than the sum of the set of *all* its direct predecessors. In fact, for strict monotonicity, let $T \uplus \{c_1, \ldots, c_k\} \longrightarrow^{\mathrm{DPD}}_R T \uplus \{c'\}$ with $c_i \prec_R c'$ for all $1 \le i \le k$. Then $W^{\mathrm{DPD}}_R(\{c_1, \ldots, c_k\}) = \sum_{1\le i\le k} w^{\mathrm{DPD}}_R(c_i) < w^{\mathrm{DPD}}_R(c) = W^{\mathrm{DPD}}_R(\{c'\})$, by definition of w^{DPD}_R, and $W^{\mathrm{DPD}}_R(T \uplus \{c_1, \ldots, c_k\}) < W^{\mathrm{DPD}}_R(T \uplus \{c'\})$.

Transitive predecessors dominance. Analogous to the DPD case, a TPD preserving weight assignment function w^{TPD}_R needs to make sure that $w^{\mathrm{TPD}}_R(c) > \sum_{c' \in C_s : c \succ_{R^+} c'} w^{\mathrm{TPD}}_R(c')$. Since, as mentioned in Section 5.3, TPD is DPD for R^+, the function $w^{\mathrm{DPD}}_{R^+}$ would suffice. However, we can avoid computing the transitive closure of R by using the following function that also only depends on the direct predecessors:

$$w^{\mathrm{TPD}}_R(c) = 1 + \textstyle\sum_{c' \in C_s : c \succ_R c'} (2 \cdot w^{\mathrm{TPD}}_R(c') - 1) \quad \text{for } c \in C_s.$$

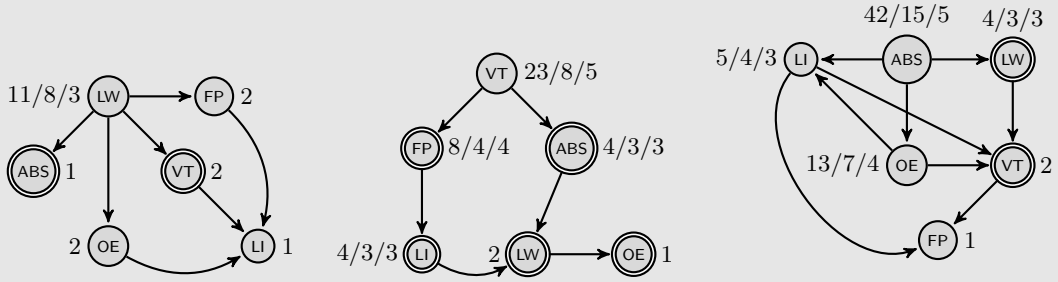
We can establish that $w^{\mathrm{TPD}}_R(c) \ge 1 + \sum_{c' \in C_s : c \succ_{R^+} c'} w^{\mathrm{TPD}}_R(c')$ for all $c \in C_s$ with this definition by induction over the number of transitive predecessors of c: If $\{c' \in C_s \mid c \succ_{R^+} c\} = \emptyset$, then

$w_R^{\mathrm{TPD}}(c) = 1$ and thus the claim holds. Let $\{c' \in C_s \mid c \succ_{R^+} c\} \neq \emptyset$ and assume that $w_R^{\mathrm{TPD}}(c') \geq 1 + \sum_{c'' \in C_s : c' \succ_{R^+} c''} w_R^{\mathrm{TPD}}(c'')$ holds for all $c' \in C_s$ such that $c \succ_R c'$. In summary, this yields $w_R^{\mathrm{TPD}}(c) \geq 1 + \sum_{c' \in C_s : c \succ_{R^+} c'} w_R^{\mathrm{TPD}}(c')$.

Theorem 1 *If $t >_R^p u$, then $W_R^p(T) < W_R^p(U)$ for $p \in \{\mathrm{SPD}, \mathrm{DPD}, \mathrm{TPD}\}$.* ∎

In particular, $>_R^p$ is asymmetric for $p \in \{\mathrm{SPD}, \mathrm{DPD}, \mathrm{TPD}\}$ since the order $<$ on the weights is asymmetric: If we would have $t >_R^p u$ and $u >_R^p t$, then $W_R^p(T) < W_R^p(U)$ and $W_R^p(U) < W_R^p(T)$ by the theorem, which is impossible.

Example—Ski-Day Planner
Figure 5.6 depicts constraint relationship graphs corresponding to the relationships established earlier. Table 5.1 shows how the assignments are evaluated using these relationships. Every user favours a different assignment. *A* indicated little interest in variety, avoiding black slopes or easy tracks and got the only assignment that does not require waiting. Similarly, we get a match for *B*'s requirements and do not force *C* to take difficult slopes. The calculated assignment winners thus make sense and show how the different graphs influence the decision process to a strong degree. Interestingly for *B*, the selected dominance property affects the preferred solution. Since $t_X^{(2)}$ only satisfies VT and violates the 5 other constraints, it is considered worse than $t_X^{(3)}$ in SPD and DPD semantics. However, as VT is the most important constraint for *B* and is only satisfied by $t_X^{(2)}$, in a TPD semantics this solution is still preferred over the others satisfying more constraints.



Graph for Skier A. Graph for Boarder B. Graph for Rookie C.

Figure 5.6: The constraint relationship graphs for each persona. Double borders indicate that this constraint was violated in the TPD-preferred assignment according to Table 5.1. Weights are printed for TPD/DPD/SPD, only one number indicates that the weights are equal for all dominance semantics.

Table 5.1: Different tours rated by different relationship graphs.

	A	*B*	*C*
$t_X^{(1)}$	**3**	8	7
$t_X^{(2)}$	9	13	16
$t_X^{(3)}$	5	**7**	**5**

SPD semantics.

	A	*B*	*C*
$t_X^{(1)}$	**3**	11	17
$t_X^{(2)}$	14	13	30
$t_X^{(3)}$	10	**10**	**5**

DPD semantics.

	A	*B*	*C*
$t_X^{(1)}$	**3**	27	44
$t_X^{(2)}$	17	**19**	65
$t_X^{(3)}$	13	25	**6**

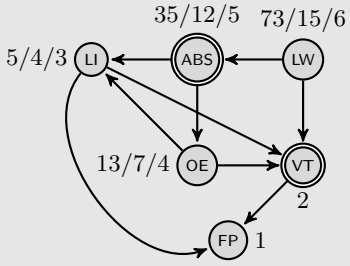
TPD semantics.

Changing Preferences In multi-agent-systems, agents change their goals depending on their perceived environmental situation (Doyle and McGeachie, 2003). Similarly, our personas may change their constraint relationships given new circumstances.

Assume, e.g., *C* has gotten enough practice such that avoiding black slopes is not as important as before—but he refuses to wait. Hence, the edge ABS $\succ_R$ LW gets inverted, making LW the most important constraint. Assignment $t_X^{(1)}$ is the only one that has a route without much waiting—and it is now favoured by *C* (see Figure 5.7).

Changing Constraints In open-world scenarios and dynamic CSPs, the set of constraints frequently changes. Assume that slopes have been evaluated for beautiful landscapes (BL) and foggy slopes (FS) can be avoided.

Given that A does not care too much about landscapes it is safe to assume that those constraints would be ranked even less important than LI. Boarder B, however, does care about BL and marks them more important than FS, FP, and ABS. Assume further that $t_X^{(1)} \models \{\mathsf{BL}, \mathsf{FS}\}$, $t_X^{(2)} \models \{\mathsf{FS}\}$ and $t_X^{(3)} \models \{\mathsf{FS}\}$. It is easy to calculate that A still ranks $t_X^{(1)} >_R^{\text{TPD}} t_X^{(3)} >_R^{\text{TPD}} t_X^{(2)}$ even though different numeric values are placed. For B the situation is different, as BL is only satisfied by $t_X^{(1)}$ which is why it would then be the preferred solution of B.



New graph for Rookie C.

	C new
$t_X^{(1)}$	**7**
$t_X^{(2)}$	19
$t_X^{(3)}$	8

SPD semantics.

	C new
$t_X^{(1)}$	**14**
$t_X^{(2)}$	39
$t_X^{(3)}$	17

DPD semantics.

	C new
$t_X^{(1)}$	**35**
$t_X^{(2)}$	127
$t_X^{(3)}$	75

TPD semantics.

Figure 5.7: After adaptation, $t_X^{(1)}$ is now preferred by rookie C

Solving the Weighted CSP

Once the weights of the individual constraints have been established, the problem presents as a normal Weighted CSP and can be solved with appropriate algorithms (Ansótegui et al., 2010; Schiex et al., 1995; Larrosa and Schiex, 2004). Alternatively, the original constraint satisfaction problem can be restated as a constraint satisfaction optimization problem. For this purpose, each constraint is reformulated as follows:

$$c_i' : c_i \vee p_i = w_R^p(c_i)$$

Alternatively, the assignment of 0 to the appropriate entry in the penalty vector can also be made explicit, depending on the solver used and, of course, on personal taste:

$$c_i' : (c_i \wedge p_i = 0) \vee (\neg c_i \wedge p_i = w_R^p(c_i))$$

This allows a constraint optimizer to either fulfil c_i or set $p_i \neq 0 \in \mathbb{N}$, the penalty for violating c_i, to the weight of the corresponding constraint. The objective is then to minimize the sum of penalties over all constraints in C with respect to the threshold k:

$$\min \sum_{c_i \in C} p_i, \text{subject to} \sum_{c_i \in C} p_i < k$$

A constraint relationship problem formulated this way can be solved with commercial or open source software, such as the IBM ILOG CPLEX Optimizer (CPLEX, 2013) which we used for a prototypical implementation. However, the approach described here is independent from the concrete language the constraints or problems are formulated in or the solver that is used.

Advantages and Extensibility

The main advantage of using a relation on constraints to express preferences instead of assigning the weights to the constraints directly is the flexibility in large and dynamic problems. If constraints have to be added during design-time or during run-time, it is not necessary to manually change the entire annotation system. Instead, only the annotations for constraints that are in a (transitive) relationship

with the newly introduced one have to be adapted, a step that is fully automated by a conversion tool, however.

Also, the mechanism can be used to check for consistency of the constraint system. If a topological sorting of the automatically calculated graph fails, the graph is not acyclic and thus the specification is inconsistent. As the cycle can be identified easily, the modeller has direct and constructive feedback about the problem and can make specific changes to correct it.

As Doyle and McGeachie (2003) observe, numerical mappings of weights to constraints need not necessarily reflect the rationales and considerations underlying such a quantitative measure. They can be the result of a long and arduous deliberation process but carry none of the information into the model. Qualitative measures, however, such as the one provided by constraint relationships purport at least some of the meaning and rationale into the model and thus allow modellers to change them later on the basis of the knowledge captured in the relationships.

If the constraint relationship system of Figure 5.3 is extended by giving another relationship $c_{14} : (co, l) \prec_R c_{15} : (co, s)$ (indicating that a corduroy jacket goes better with leather shoes than it does with sneakers), the graph is extended with an unconnected subgraph consisting of the nodes c_{14} and c_{15}. If such a subgraph exists, the same algorithm as above is applied. Using the proposed weighting function, this results in the annotations c_{14} with a weight of 2 and c_{15} with a weight of 1. Therefore, implicit relationships between the other constraints in the system are introduced. As the weight of c_{14} is less than the weight of c_1, e.g., the latter constraint is considered more important than the former. Since there is no effect of the new relationships on the weights for c_1 and others and thus not on the upstream weight for c_7, the semantics of these implicit relationships are different. However, as there is no explicit relationship, this notion makes intuitive sense. If the modeler wanted a relationship between c_{14} and c_7, it could be made explicit by, e.g., introducing $c_7 \prec_R c_{14}$. In any case, a modeler has to be mindful of such implicitly defined relationships and make careful and conscious decisions about them. As a direct graphical representation of the relationships is provided, such cases are easy to detect and resolve if necessary.

5.5 Discussion and Related Work

The distinction between hard and soft constraints and the preferences over soft constraints established by constraint relationships solve two problems:

1. They allow directed relaxation of over-constrained problems while maintaining irrevocable restrictions.
2. They provide the modeller with a simple to use tool to express preferences over constraints. Existing relationships can be easily extended without the need to recalculate weights or penalties and the problem can automatically be translated into a form usable with constraint optimization software.

The basic technique to deal with over-constrained problems (Jampel, 1996) is constraint relaxation. For most constraint satisfaction problems, this means omitting certain, less important constraints. This is usually done with a form of partial constraint satisfaction or a derivative thereof as discussed, e.g., in (Guesgen and Hertzberg, 1992) and further below. Over the years, a number of formalisms for soft constraints have been proposed. Constraint hierarchies with locally-predicate-better comparators can be fully emulated with constraint relationships. Valued and semiring-based constraint satisfaction provide a formal basis with which soft constraint problems of different color can be described. Special measures need to be taken if the information about soft constraints is incomplete. Conditional preferences provide the possibility to express statements of the form "if a, prefer b over c" and are formalised with so-called CP-nets. All mentioned approaches are discussed in the following and related to the approach proposed here.

Constraint Relaxation

Another approach that is worth mentioning, however, is pursued when constraints are used to specify numerical optimization problems that are solved, e.g., with linear programming or mixed-integer programming techniques. In these cases, instead of dropping individual constraints, either the domains of variables are extended or the coefficient of constraints are changed. This is usually done by introducing *slack variables* that extend the range of values allowed by constraints. The goal is then to minimize the slack and thus the error introduced.

Although this course of action violates the constraints originally defined to secure a correct valuation of the variables, it is often advantageous to find a good engineering solution (Zhang and Irving, 1993) or to diagnose a system. The constraints that are changed can give important hints as to why there is no feasible solution and it is often possible to allow solutions with a small error, as in many engineering applications the constraints contain a generous safety margin.

The main drawback of this kind of constraint relaxation is that it is undirected. The algorithms choose the constraints autonomously and there is no notion of preference. This can lead to solutions that are utterly unrealistic or highly undesirable. The combination of an approach such as ours with this kind of constraint relaxation might yield more robust solutions that are more realistic and thus usable. This, however, is left as future work.

Constraint Hierarchies

Borning et al. (1992) introduce *Constraint Hierarchies*, in which required and preferential constraints are specified separately. Required constraints form the topmost level of the hierarchy, while preferential constraints can be arranged in an arbitrary number of levels where each level dominates the one below completely. To find the best solutions, the authors introduce several comparators. Local comparators consider each of the constraints individually and return the solution in which the error of one constraint is strictly less than in all other solutions and all other constraints yield at least the same error. Global comparators use an additional combining function which can be used to, e.g., incorporate weights of the constraints into the comparison of different solutions.

Hard and soft constraints can be represented as constraint hierarchies in two ways:

1. as a two-level hierarchy where hard constraints are required and soft constraints are preferential.
2. as a hierarchy with separate levels for constraints with the same weight.

In the first case, to achieve the same semantics as the one described above, the weights for the non-required level are assigned with the graph approach presented above and a weighted-sum-predicate-better comparator is used. This comparator uses a predicate comparator, i.e., a constraint is either fulfilled (error of 0) or not (error of 1) and a summation of product of the errors and the respective weight of the constraint.

Borning et al. argue against such an approach (Section 2.3 of Borning et al., 1992). They offer three reasons against it. First, when preferential constraints are all part of the same level of the hierarchy, expectations of the user or designer may be violated when constraints with little weight hold while ones with high weight do not. Second, the local comparators do not work with weights but only on individual constraints and can thus not be used in such a scenario. Finally, the algorithms for solving constraint hierarchies proposed by the authors work better when levels completely dominate the ones below.

We believe that these arguments are not valid under all circumstances. The original work of Borning et al. was focused on user interface layout, especially geometric layout which is basically a form of computer-aided design in which the user creates geometric forms that are combined from simple primitives and connected with constraints. In such a case it might indeed be relevant that a number of soft constraints are disregarded before a stronger one no longer holds as the geometric figure might fall apart otherwise. However, in systems such as the one presented here, such a requirement does not exist or not to the same extent. Additionally—depending on the dominance properties, of course—the way constraint relationships are weighted would potentially require a fairly large number of constraints must be violated before stronger constraints will be disregarded. In cases in which the use of local comparators is not required and global comparators can be used without restrictions or are even required due to the system specification, there is no reason to hold on to the possibility to use them. Finally, the transformation of the constraint relationships into a CSOP allows the use of efficient algorithms for this problem. In conclusion, although the reservations of Borning et al. might be valid in some circumstances, the decision to use a two-level approach might still make sense in cases in which system requirements allow the violation of higher weighted constraints, force global comparators and the use of a CSOP solver is possible.

In the second case (using a hierarchy with a separate level for constraints with the same weight), the original intention of Borning et al. is preserved much better than in the first case. The benefit of using constraint relationships is an automated creation of the hierarchy levels. Such a conversion adheres to the definitions of dominance put forward in (Borning et al., 1987).

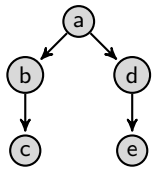


Figure 5.8: A set of constraint relationships not expressible in LPB-hierarchies.

Encoding Constraint Hierarchies with Constraint Relationships Constraint hierarchies offer different *comparators* (Borning et al., 1992) to discriminate solutions based on their satisfaction degree of constraints at different levels given by an error function and additional weights for constraints. Comparators are divided into *locally better* and *globally better*. Locally better compares based on the error functions only, whereas globally better predicates also take constraint-specific weights into account.

Error functions are either *predicate* functions, i.e., $e(c,t) = 0$ if $t \models c$, and 1 otherwise; or *metric* functions that give a continuous degree of violation (e.g., for $c \triangleq (X = Y)$, $e(c,t)$ could return the difference between the valuations of the variables X and Y in t). Since our approach is not concerned with metric error functions or user-defined weights, we restrict ourselves to comparison with *locally predicate better* (LPB) and show that these hierarchies can be encoded in constraint relationships. We then show that constraint relationships generalise LPB-hierarchies by providing an example that cannot be expressed by them.

First, consider the definition of LPB given by a constraint hierarchy $H = \{H_0, \ldots, H_n\}$. The operator $>_{LPB}$ compares two solutions t and u (the constraints in H_0 are taken to be the hard constraints) and $t >_{LPB} u$ should be read as "t is better than u"; it is defined by

$$t >_{LPB} u \leftrightarrow \exists k > 0 : (\forall i \in 1..k-1 : \forall c \in H_i : e(c,t) = e(c,u)) \wedge (\forall c \in H_k : e(c,t) \leq e(c,u)) \wedge (\exists c \in H_k : e(c,t) < e(c,u))$$

Our encoding of the constraint hierarchy H in constraint relationships R_H is defined as follows: $C_h = H_0$, $C_s = \bigcup_{i \in 1..n} H_i$, and

$$c \succ_{R_H} c' \leftrightarrow c \in H_i \wedge c' \in H_{i+1} \ .$$

We write T_i for all constraints in hierarchy level i that are violated by an assignment t, i.e., $T_i = \{c \in H_i \mid t \not\models c\}$ and, analogously, U_i for an assignment u; we abbreviate $\bigcup_{k \leq i \leq l} T_i$ by $T_{k..l}$.

Theorem 2 *If $t >_{LPB} u$, then $T \longrightarrow^{\mathrm{TPD}}_{R_H} U$.*

Proof Observe that $e(c,t) < e(c,u)$ iff $t \models c \wedge u \not\models c$; and $e(c,t) \leq e(c,u)$ iff $u \models c \rightarrow t \models c$.

Let $t >_{LPB} u$; we have to show that T is worsened to U by application of TPD-rules. Let $k > 0$ be such that (*) $\forall i \in 1..k-1 : \forall c_i \in H_i : t \models c_i \leftrightarrow u \models c_i$, (**) $\forall c_k \in H_k : u \models c_k \rightarrow t \models c_k$, and let $c \in H_k$ such that $t \models c \wedge u \not\models c$. By (*) we have that $T_{1..k-1} = U_{1..k-1}$. Furthermore, $T_{1..k} \subseteq U_{1..k} \setminus \{c\}$ since $T_{1..k-1} = U_{1..k-1}$, $\forall c_k \in H_k : t \not\models c_k \rightarrow u \not\models c_k$ by (**), and $c \notin T_{1..k}$. In particular, $T_{1..k} \subseteq U_{1..k} \setminus \{c\} \subseteq U_{1..n} \setminus \{c\}$. If $T_{1..k} = U_{1..n} \setminus \{c\}$, then $T = T_{1..k} \uplus T_{k+1..n} \longrightarrow^{\mathrm{TPD}}_{R_H} T_{1..k} \uplus \{c\} = U$ by (TPD), since all constraints in $T_{k+1..n}$ are transitively dominated by $c \in H_k$. If $T_{1..k} \subsetneq U_{1..n} \setminus \{c\}$, then $T_{1..k} \longrightarrow^{\mathrm{TPD}}_{R_H} U_{1..n} \setminus \{c\}$ by (W1'); applying rule (TPD) in (W2), we again have $T = T_{1..k} \uplus T_{k+1..n} \longrightarrow^{\mathrm{TPD}}_{R_H} (U_{1..n} \setminus \{c\}) \uplus \{c\} = U$. ∎

Conversely, Fig. 5.8 shows a constraint relationship problem that is not expressible in LPB hierarchies. Let $H : \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{e}\} \rightarrow \mathbb{N} \setminus \{0\}$ be a mapping from the constraints to their respective hierarchy levels. We consider solutions that only satisfy *one* constraint and violate all others and write a for "a solution satisfying only a". We show that every admissible choice of H introduces too much ordering: The constraint relationships require a to be better than b which in turn should be better than c, thus we have to have $H(\mathsf{a}) < H(\mathsf{b}) < H(\mathsf{c})$. Since we expect a to be better than d as well, but require b and d to be incomparable, $H(\mathsf{d})$ has to be equal to $H(\mathsf{b})$. Similarly, $H(\mathsf{e})$ has to be $H(\mathsf{c})$ as e and c should be incomparable. But then b would be better than e, a relation that is explicitly not modeled in the underlying constraint relationships.

Valued and Semiring-Based Constraint Satisfaction

There have been a number of approaches that try to consolidate different takes on soft constraints, including valued constraint satisfaction (VCSP) (Schiex et al., 1995) and semiring-based constraints satisfaction problems (SCSP) (Bistarelli et al., 1997). SCSPs and VCSPs are compatible with each other and it is possible to pass from one to the other (Bistarelli et al., 1999).

In VCSPs, a valuation structure in the form of a totally ordered, commutative monoid is given. It allows to assign a valuation to each constraint and to formulate preference hierarchies. However, as the constraints are annotated directly, it becomes very difficult to handle relationships between constraints. Each relationship has to be incorporated when assigning the valuations and additional constraint relationships might require the change of many valuations to accommodate the new relationship structure. Our approach in comparison does not require such a backtracking as it allows the definitions of the relationships directly and explicitly.

SCSPs define preferences over the values of variables. A soft constraint can be defined as a pair $\langle def, con \rangle$ in which $con \subseteq V$ is a subset of the variables V and $def : D^{|con|} \rightarrow A$ is a function that maps each of the values of the variables in con from the domain D to a value of the set A which is the carrier of a c-semiring (Gelain et al., 2007). Depending on the problem, the preferences can be represented in different semirings. Classical weighted CSPs, in which the preference is a real number, can be expressed as the following semiring:

$$S_{WCSP} := \langle \mathbb{R}^+ \cup +\infty, min, +, +\infty, 0 \rangle$$

The c-semiring for k-weighted CSPs, used here to express constraint relationships has been given in Section 5.2. We are currently investigating the definition of a semiring that captures constraint relationships directly.

Incomplete Soft Constraint Problems

The semiring-formalism has been extended to accommodate incomplete information about the preferences or unspecified values in the domain (Gelain et al., 2007). It assumes that the modeller or the user of the model states preferences explicitly, e.g., by assigning weights or a preference value. The preference values imply an ordering of the constraints. The optimal solution to a problem where the order is not total is one in which the global preference is not dominated. Such a solution can not always be found since preferences might be missing to decide on the optimal solution and it is only possible to calculate solution candidates.

The order implied by the preference values in such an incomplete soft constraint problem can also be represented as a number of constraint relationships. The relationships make the order explicit without the need to state a preference value. This significantly simplifies the modelling process and yields a more concise model without implicit relations between constraints that are introduced by the assignment of preference values. On the other hand, it forces the modeller to make every relation that should be regarded explicit.

The solution algorithm proposed in (Gelain et al., 2007) computes a set of solution candidates and elicits additional preferences from the user if necessary. This constitutes an important difference to our approach: a constraint optimizer would output one solution with minimal penalty with the given relationships.

Conditional Preference Statements in Decision Theory

Apart from constraint problems, decision theory constitutes another field in which the expression of preferences is important. In (Boutilier et al., 1997, 1999, 2004), an approach to capture conditional qualitative preferences ("I prefer red wine over white wine") is described that has some similarities to the one described in this paper. This decision theoretic approach can be translated to a soft constraint problem (Domshlak et al., 2003), albeit with some restrictions.

In decision theory, one is interested in capturing a user's preference, especially eliciting them in an interactive process that presents a user with the minimal set of choices that guide the search for a solution in an optimal way. Therefore, (Boutilier et al., 1999) proposes preference elicitation for a qualitative ranking of outcomes, i.e., the user is presented with different choices that can be ranked according to their preference. This process also allows the formulation of conditional preferences ("If I wear a red shirt, I want to wear grey pants"). However, the process is still quite involved. The user

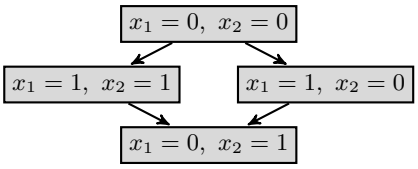


Figure 5.9: Desired solution order, expressed as a CP-Net. The solution $x_1 = 0,\ x_2 = 0$ should be the best.

has to define parent features for each feature (corresponding to a variable) and preferences can only be expressed between parent features and their child features. Then, the user has to specify preferences over all feature *values* for all parent feature *values*. These information can be visualized with CP-nets, a graphical representation of the relations between features and their parents which are annotated with tables that express the preferences over the values of parent features and features. In principle, the preference relations are user-specific constraints. Finding a solution – a problem that seems to be very suitable for constraint solvers – is performed by flipping values of the outcomes according to certain heuristics that do not require backtracking in most cases and are hence very efficient.

CP-nets (Boutilier et al., 1997) specify total orders over the domain of a variable depending on an assignment to other variables in a conditional preference table. Concretely, a preference statement for a variable y is written as $x_1 = d_1, \ldots, x_n = d_n : y = w_1 \succ \cdots \succ y = w_k$ where $x_1, \ldots, x_n$ are the *parent* variables of y and $w_1, \ldots, w_k$ are all domain values of y given in a total order $\succ$. Such an order needs to be specified for all assignments to $x_1, \ldots, x_n$. A preference statement should be interpreted as "Given that $x_1 = d_1, \ldots, x_n = d_n$, all other variables being equally assigned, prefer a solution that assigns w_i to y over one that assigns w_j to y iff $i > j$" which is the *ceteris paribus* assumption. The change of value for y from w_i to w_j is then called a "worsening flip". A complete assignment t to the variables of a CP-net is preferred to another one, say t', if t' can be obtained from t via a sequence of worsening flips (Rossi et al., 2006).

On the one hand, the induced "better-as" relation on assignments need not be a partial order since cycles may arise (Boutilier et al., 2004). By contrast, constraint relationships always lead to a partial order $>_R^p$ on assignments. On the other hand, CP-nets cannot express all partial orders on assignments (Rossi et al., 2008). Consider the minimal example depicted in Fig. 5.9: $X = \{x_1, x_2\}$, $D_1 = D_2 = \{0, 1\}$. The proposed solution order cannot be expressed in CP-nets since $x_1 = 1,\ x_2 = 0$ and $x_1 = 1,\ x_2 = 1$ differ only by the assignment of x_2 and have to be comparable because of the total order requirement and *ceteris paribus* semantics in CP-nets. But this solution ordering is easily expressible in constraint relationships defining a constraint for each possible assignment.

The preference statements encoded in a CP-net can be translated into a soft constraint problem (Domshlak et al., 2003) that captures the original preference order. The approximation works only for discrete, finite domains. The authors claim that the approach allows for the combination of conditional preferences and soft constraint problems but never actually show how an existing problem can be extended with either. The main area of application is preference elicitation directly from a user and solution of the problem with standard constraint solving techniques.

Apart from the mentioned differences, our approach also differs from CP-nets and their translation to soft constraints in several important aspects:

- Most importantly, constraint relationships are not limited to conditional preferences that depend on a certain structure of the variables. CP-nets impose a parent-child relation on the variables and require the value of children to be conditional on the values of the parents. While this allows efficient algorithms to find optimal valuations, it is a rather strict prerequisite that such a structure has to be imposed. While constraint relationships can not express such preferences, they also do not require any structure on the variables but rather on the constraints. Depending on the problem, either of these approaches can be more suitable.
- We define preferences over constraints, not on the outcomes. Although the constraints used as examples in this paper can be interpreted as outcomes as well (extensional constraints), our approach is not dependent on the way constraints are formulated and is applicable to intensional constraints as well.
- The target audience of our approach are modellers of complex constraint systems, not end users. A modeller needs different tools and feedback mechanisms which are provided in the form of

annotated DAGs—that show the relations of constraints instead of the relations of variables—and easy integration with commercial constraint solvers which is achieved with the translation outlined in this paper.

- Our approach is not limited to variables with finite, discrete domains. Again, as constraint relationships are independent of the form of the constraints and can be used in any circumstances, constraints and variables can be arbitrarily complex.

Thus, the two frameworks are incomparable regarding the solution order. An extension, however, of the constraint relationship approach with conditional statements as in CP-nets might turn out to be fruitful.

Chapter Summary and Outlook

This chapter introduced the Restore Invariant Approach for the specification of correct system behaviour as well as constraint relationships as a means to specify soft constraints and preferences over them. Constraint relationships impose a partial order over the constraints, depending on the dominance property used. The different dominance properties can also be useful to determine how much more important individual soft constraints are than others. The transformation into a k-weighted constraint satisfaction problem allows checking the relationships for consistency and make it possible to use a standard solver to compare solutions that violate different soft constraints and find the ones with a minimal violation. Existing soft constraint formalisms can either be simulated with constraint relationships or are complementary to the approach. Constraint relationships are used in the specification of behavioural corridors according to the RIA to denote "good" or "optimal" system behaviour as described in Chapter 6 as well as in the specification of adaptation problems as shown in Chapter 9.

Chapter 6

Monitoring System Behaviour with Soft Behavioural Corridors

Summary. In this chapter, we show how the Restore Invariant Approach and its extension with constraint relationships can be used in a self-adaptive system to monitor the behaviour of the system and detect a violation of the behavioural corridor. For this purpose, a flexible Observer/Controller architecture is used that allows individual controllers to register with different observers in a publish/subscribe scheme. Thus, different controllers can be used to react to the violation of different constraints, allowing for a staged reaction to different errors and events. In addition, constraints, observers, and controllers can be added to the system at runtime, allowing the adaptation of the system to new circumstances and contributing to the goal of creating long-lived systems that are robust to changes in the environment and in their requirements. Finally, black-box and white-box monitoring are compared and monitoring in hierarchical systems is illustrated. The concepts developed in this chapter are set side by side with approaches from the literature.

Publication. The concepts in this chapter are an extension of the work published in (Steghöfer et al., 2013c) and (Eberhardinger et al., 2013).

The Restore Invariant Approach (RIA) allows the specification of correct system behaviour with constraints, especially of emergent behaviour that was unexpected at runtime. Its extension with soft constraints, as introduced in the previous chapter, allows the expression of qualitative differences between states of the system. In order to use the RIA to observe functional correctness at runtime and to start a reconfiguration in case the invariant is violated, a monitoring infrastructure has to be in place that gathers the current system state, observes the constraints, and informs one or several controllers if constraint violations occur. Such an infrastructure, together with an appropriate reconfiguration algorithm (cf. Part IV), can ensure functional correctness at runtime (Nafz et al., 2011).

The Observer/Controller (O/C) architectural pattern introduced in Chapter 3 is the basis for this monitoring infrastructure. It includes an *observer*, the functional element responsible for monitoring the system under observation and control and analysing the information. While the O/C architecture is used in many Organic Computing applications and—in form of the MAPE-cycle or other feedback loops—in many other adaptive systems as well, most of the implementations are ad-hoc and aimed at a specific system. Our goal is to define a generic O/C model that can be used in code transformation (cf. Chapter 7) and that is applicable to as many systems as possible. If it can't be applied directly, it should at least be adaptable to a concrete system while maintaining most of the benefits of a well thought-trough architecture. The RIA serves as the basis for such an architecture and for the specification of the system behaviour.

In this chapter, we develop a monitoring infrastructure (see Section 6.1) that can be used as a basis for customised instantiations of O/Cs in large-scale open self-organising systems based on the RIA. We show the interactions between the agents and the O/C, as well as the internal control flow between observer and controller. Different types of soft constraints are of course supported by the instantiation (see Section 6.2) and it is possible to observe both hard and soft constraints at the same time and react to their violation independently. Whenever a hard constraint is violated, a new stable system state can be achieved with positive feedback (cf. Chapter 9), while the violation of a soft constraint can

either cause negative feedback (cf. Chapter 8) to maintain the existing stable state or start a "silent" reconfiguration in the background that allows the system to calculate a new configuration that fulfils the soft constraint while maintaining its functionality, as described in Section 6.3. Advanced topics, such as black-box monitoring in which it is not possible to access the agents' internal states is discussed in Section 6.4 and monitoring in systems with hierarchical structures is discussed in Section 6.5. Finally, the monitoring approach introduced in this chapter is compared to other work in the literature and is classified with respect to the related work in Section 6.6.

6.1 Operationalisation of the Restore Invariant Approach in an Observer/Controller Architecture

To be able to monitor a system appropriately it is necessary to evaluate the system invariant INV in every step $s_i \in S$ of the system trace π. In the context of the RIA it is particularly interesting to know if INV does not hold in a specific step s_i since an appropriate action has to be performed in this case. If INV holds, no action is required. Since the system invariant INV is composed of constraints $\phi \in \Phi$, their validity is fundamental for the validity of INV. The relation between INV and ϕ is as follows:

$$\forall s_i \in S : \bigwedge_{\phi \in \Phi} \phi(s_i) \rightarrow INV(s_i) \tag{6.1}$$

where $\phi(s_i)$ is the evaluation of the constraint ϕ in the state s_i of a system trace π and $INV(s_i)$ is the evaluation of the system invariant INV in the state s_i of the trace π. Based on the conjunctive form of Eq. (6.1), the violation of INV is the consequence of a violation of one of the constraints:

$$\forall s_i \in S : \exists \phi \in \Phi : \neg\phi(s_i) \rightarrow \neg INV(s_i) \tag{6.2}$$

Therefore, $\neg\phi(s_i)$ implies that the invariant is violated and a reaction of the system is required. Thus, it is sufficient to monitor each constraint $\phi \in \Phi$ separately and react to any violation of a single constraint ϕ. This alleviates the need to gain a global view of the system to monitor a violation of INV in the RIA, a fact that not only makes monitoring much easier to implement but also leads to improvements in scalability and flexibility of the monitoring approach when compared with other concepts from the literature (cf. Section 6.6).

The constraints that are part of the system invariant are based on the functional and non-functional system requirements that constitute the *correct* behaviour of the system. If any of these constraints is violated, the system does not fulfil these requirements and an appropriate action has to be taken. Issues of functional correctness as well as reliability or performance can be formulated this way.

Hard and soft constraint

The constraints that are part of the system invariant INV must not be violated in the productive system. Therefore, these are *hard* constraints. A violation of those hard constraints leads to a reaction, where the productive state is interrupted, the system is transitioned into a quiescent state—a state ensuring that the system does not impede the reconfiguration (Nafz et al., 2013)—and the system is reconfigured. For the violation of different constraints, different reactions can be appropriate.

It is often necessary, however, to describe *desired* system behaviour. If the system does not show this desired behaviour but still exhibits correct behaviour, it might not be necessary to react immediately. Therefore, requirements describing desired behaviour must be expressed as constraints that are not part of the system invariant, the so-called *soft* constraints. They define a narrower "corridor within the corridor", that, e.g., expresses the optimal behaviour of the system (cf., e.g., Quinton and Ernst, 2012).

Soft constraints can be monitored with exactly the same monitoring infrastructure as hard constraints. Depending on the way they are formulated, they can serve different purposes. If a soft constraint (e.g., $x < 10$) has a stronger condition than a corresponding hard constraints (e.g., $x < 15$) they can a) be considered optimization constraints but can also b) indicate that a system might approach the corridor of correct behaviour. If soft constraints are aggregated in hard constraints as in the example below, they are used to describe a desired state on two or more distinct variables with a hard restriction on their combination. If soft constraints indicate optimization criteria or a possible system development towards the corridor's boundaries, it might still be a good idea to react to their violation. In contrast to violations of hard constraints, however, such a reaction will not switch the system to a quiescent state in which the productive behaviour is suspended, but will work in the background.

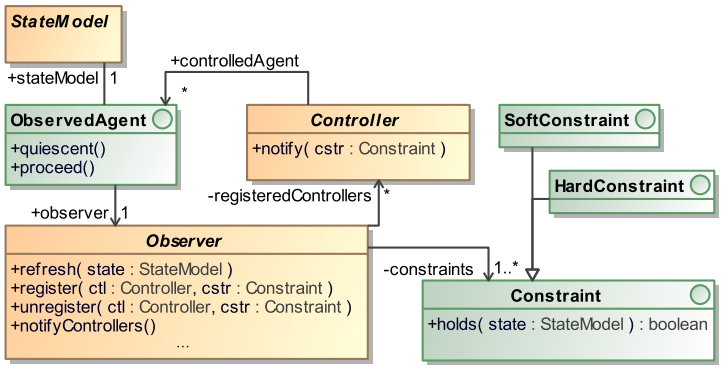


Figure 6.1: Simplified generic Observer/Controller model as used in the transformation, specified as a UML class diagram.

Example: Simple constraint scenario. Consider an agent A with three internal numerical variables, namely x, y and z, which describe its state. The value of x may be never greater than 100 or a failure will occur. Furthermore, the values of y and z should be less than 200 so that the agent works economically. Also, no system state is allowed where y and z are both greater than 200 at the same time. This textual description can be interpreted as follows:

```
context A inv c1: x<=100
context A soft c3: z<200
```

```
context A soft c2: y<200
context A inv c4: c2 or c3
```

Thus, *c2* and *c3* are soft constraints which do not affect the validity of the system invariant. These constraints can be monitored in order to optimise the system. Constraints *c1* and *c4* are hard constraints and must hold in every productive state where *c4* is a hard constraint over a set of two soft constraints. This shows that a set of soft constraints can be aggregated within a hard constraint. Please note that the **soft** keyword is a proprietary extension of standard OCL to denote a constraint that is not part of the invariant.

An Observer/Controller Model for the Restore Invariant Approach

Observer/Controller

We want to embed the use of RIA and an appropriate monitoring infrastructure into large-scale open self-organising systems. To achieve this aim, we specify a generic Observer/Controller model, as depicted in Figure 6.1, that provides the structure for the observer models and is based on the publish/subscribe pattern (Gamma et al., 1995). An `ObservedAgent` has an association to an `Observer`. The `Observer` in turn holds a list of registered `Controller`s. `Controller`s subscribe at the `Observer` to be notified in case a `Constraint` is violated. `Constraint` can either be `HardConstraint`s—meaning that their violation implies a violation of the behavioural corridor, or `SoftConstraint`s—meaning that their violation impedes the optimal behaviour of the system.

Observer vs. Monitor
While we are concerned with observing the system and thus speak of "observers", the literature often uses the term "monitor" instead. We follow Schmeck et al. (2010) and Richter et al. (2006), who define monitoring as part of the observer that collects and structures raw data. This same view is employed in the MAPE-cycle as well (Sterritt et al., 2005; White et al., 2006). Leucker and Schallhart (2009) define monitoring more broadly, including analysis and evaluation of collected data. They also include checking the data for correctness, similar to Kim et al. (1999) and Calinescu et al. (2012).

The relevant interaction between these classes is depicted in Figure 6.2. Whenever an `ObservedAgent` registers a change (basically, a transition in SYS from s_i to s_{i+1}), it informs its `Observer` by sending the state model with the updated information. The `Observer` then updates its state model for s_{i+1} of the agent and evaluates all `Constraints` $\phi \in \Phi$. If one of them evaluates to false, i.e., $\neg\phi(s_{i+1})$, the `Observer` informs all `Controller`s which are registered for this constraint. Each `Controller` decides whether or not to enact changes in the system. In order to do this, it can interrupt the productive state of the `ObservedAgent` by calling the `quiescent()` method. After any changes have been performed, the `Controller` sends a signal to the `ObservedAgent` to return to the productive state, by calling the `proceed()` method. Depending

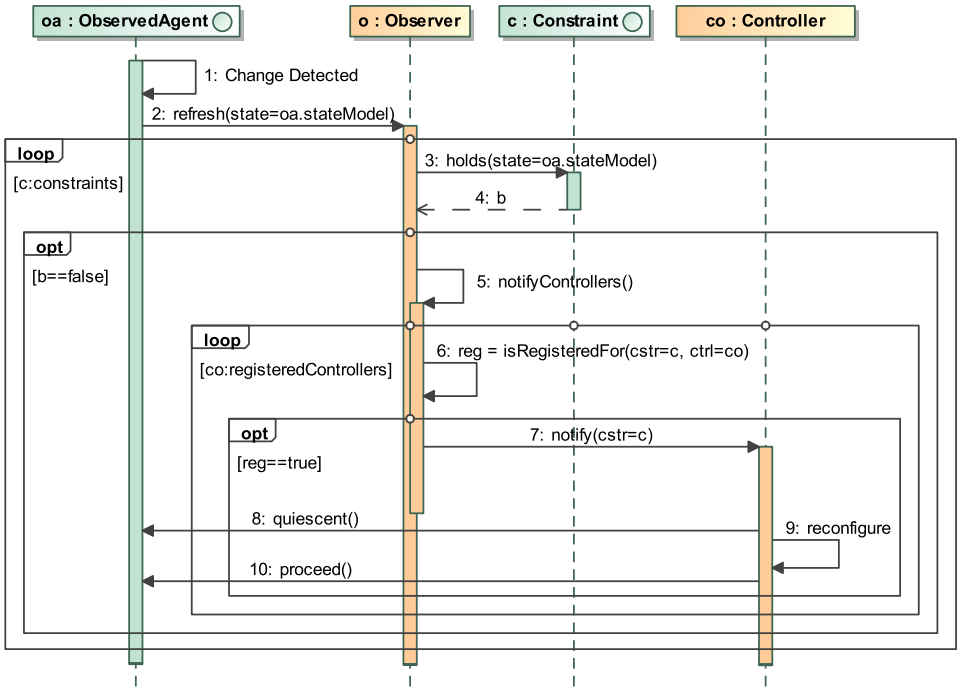


Figure 6.2: A detailed sequence diagram showing the interactions between the elements of the generic Observer/Controller model. An agent sends its current state model to the observer whenever a change in the state has been detected. The observer then checks all constraints. If a constraint no longer holds, the controllers registered for it are activated and can reconfigure the system. The controllers only transition the system into a quiescent state if the violated constraint is a hard constraint.

on the reaction to the constraint violation, the `Controller` might reconfigure several agents or ask other controllers to participate in the reaction. In such a case, the global quiescent state is left when all `Controller`s have told the `ObservedAgent`s to leave the quiescent state and to continue with their productive work.

The generic O/C model has been designed to provide a maximum of flexibility at runtime. Since it is based on the publish/subscribe paradigm, controllers can easily be added at runtime. This allows changes to the adaptation mechanisms in a running system, making it possible to replace adaptation mechanisms with newer versions over the lifetime of the system. The observer also provides methods to add new constraints at runtime, making it possible to introduce new requirements that need to be observed. When combined, these features give a high degree of adaptability in the Observer/Controller part of the system while ensuring separation between the agent and the O/C infrastructure. Unfortunately, the model does not give the same flexibility to changes in the observed state data. If a new constraint is defined on a parameter of the observed agent that is currently not part of the state model, the agent and the state model implementation have to be changed.

Alternative Observer/Controller Models without Observer Notification

The generic O/C model presented here requires the agent to willingly disclose the information required by the observer to check the constraints. This behaviour can be codified in the interface specification for agents willing to participate in an open self-organising system. However, it is worthwhile to consider alternative approaches in which such a requirement must not be defined. There is a strong connection to black-box monitoring, discussed in Section 6.4, but in the following, we only consider white-box monitoring in which it is possible to get the necessary data from the agents directly.

The model presented above can be characterised as a *push-model* since the agent pushes changed information to the observer. Such models are well-suited if the frequency of changes is relatively low. If a part of the state changes often, transferring the entire state model might be considered overhead, especially if the part that changes is only a small element within the state model. Especially in message-based multi-agent system, the computational cost of serialising, transmitting, and deserialising instances of `StateModel` often can accumulate and make pushing the state uneconomical.

An alternative observer could use a *pull-model* that accesses the state data directly by repeatedly querying the relevant information. The frequency of the queries has to be adapted to the expected speed with which the data changes and the amount of time needed for each such interaction. Intuitively, if issuing the query, receiving the data, and incorporating it into the state model—which is now maintained by the observer—takes 50 ms, it makes no sense to query the information every 30 ms. Such an alternative monitoring approach incurs a triggering problem: while the O/C architecture above assumes that the agent voluntarily informs the observer about changes in its state, the alternative observer has to detect observable changes or interactions itself. While the former observer can thus be inactive as long as it is not informed about state changes, the latter observer has to be active constantly. If the observer misses a change in the agent state, the system might leave the corridor of correct behaviour for a short time without the observer being able to detect the violation. Such transient errors are, however, often without consequence and can thus be ignored.

The triggering problem can be mitigated by a number of technical solutions. Observers can, e.g., run as separate threads and pull the data with a high sampling frequency. Alternatively, they can make use of the abilities of the implementation framework. Jadex, e.g., a multi-agent system that has been used to implement an adaptive production cell case study (Nafz et al., 2009a), allows specifying conditions on the agents' data. A condition corresponds to a constraint and its violation causes an event the agent can react to by, e.g., informing the controller. Such a solution is discussed in (Seebach et al., 2011). The authors also introduce separate controller agents (called "reconfiguration agents" there) that run concurrently with the functional system and can reconfigure the system while the so called "base agent" continues to work. Such an architecture is very much suitable for use with silent reconfigurations (see below) and for long running adaptation processes.

No matter how a pull model is actually implemented, it is necessary to be able to access the information required to evaluate the constraints. This means that some kind of interface must be codified in the requirements for agents that participate in the open system if they must be part of a feedback loop. An observer implementing a pull-model can then act as a wrapper around the agents and embed them in the adaptation processes of the system.

6.2 Monitoring Soft Constraints

As outlined above, soft constraints can describe preferred system states, adding a level of optimality to the corridor of correct behaviour. Soft constraints can be defined as separate, observable constraints. If a controller registers for their violations, this controller can react to their violations by performing an adaptation that does not require the system to be transitioned into a quiescent state. Insofar, the observer model and the control flow described in the previous section change only in calls to `quiescent()` and `proceed()` when soft constraints are used. Soft constraints can also be incorporated into hard constraints, either by imposing a limit on the weight of soft constraints violated at the same time or by imposing a limit on the number of times a soft constraint may be violated within a given time span. Both possibilities have to be handled accordingly by the observer model and are discussed further below.

The example in Section 6.1 introduces another way of dealing with soft constraints. The constraint `c4` contains a logical expression over two soft constraints, `c2` and `c3`.

```
context A inv c4: c2 or c3
```

A generic observer model to accommodate such complex logical expressions would be very hard to define and to transform since all possible combinations of operators would have to be taken into consideration. A simpler solution is to expand the soft constraints directly in `c4` during the transformation process illustrated in Chapter 7. Effectively, `c4` then becomes:

```
context A inv c4: (y<200) or (z<200)
```

Soft Constraints specified with Constraint Relationships

Complex systems of constraints make it necessary to express preferences over constraints as discussed in Chapter 5. These preferences can be specified as *constraint relationships*. If this is the case, the constraints are assigned a weight, or *penalty*. The penalties can be used in monitoring by limiting the total sum of penalties of violated constraints to a maximum by introducing a hard constraint that

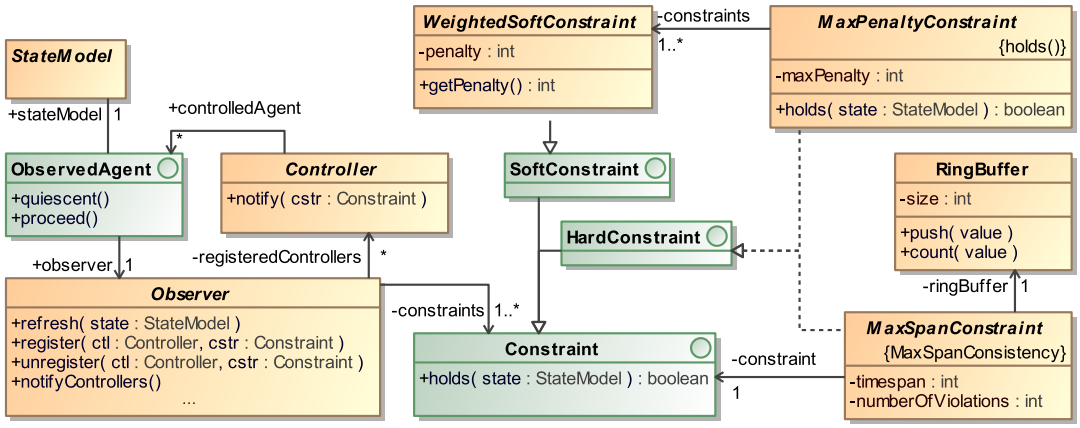


Figure 6.3: The extended Observer/Controller model, containing the elements from Figure 6.1 as well as the class `MaxPenaltyConstraint` and `MaxSpanConstraint`.

imposes such a limit. A violation of such a hard constraint over the sum of the penalties can then cause a reconfiguration in case too many soft constraints are violated at the same time. The specification of soft constraints and constraint relationships during the requirements engineering process and the transformation into monitoring constraints is described in Section 7.3.

As Figure 6.3 shows, a `MaxPenaltyConstraint` has an association to an arbitrary number of `WeightedSoftConstraints` that carry a penalty. The `holds()` method of the `MaxPenaltyConstraint` calls `holds()` on all constraints. If this method returns false, the penalty—accessible by calling `WeightedSoftConstraint.getPenalty()`—is added to the overall penalty. If the overall penalty exceeds `maxPenalty`, the `MaxPenaltyConstraint` is violated and returns false. In OCL, this can be expressed as follows:

```
context MaxPenaltyConstraint def holds(): Boolean =
let violatedConstraints : Set(WeightedSoftConstraint) =
  self.constraints->select(c: WeightedSoftConstraint | not c.holds(state))
in
  violatedConstraints->iterate(c: WeightedSoftConstraint,
                               aggregatedPenalty = 0,
                               aggregatedPenalty + c.getPenalty())
    <= self.maxPenalty
```

Listing 6.1: OCL expression for evaluation of a MaxPenalty constraint

MaxSPAN Constraints

As in the work described here and in the next chapter, Quinton and Ernst (2012) use an analysis of the system requirements and the system environment as the basis for the formulation of constraints. The authors regard real-time systems exclusively and thus specifically focus on timing behaviour as part of the constraints. These requirements are captured by *weak* and *hard* constraints, comparable to the soft and hard constraints introduced in this thesis. In the context of Quinton and Ernst (2012), however, weak constraints may only be violated a certain number of times within a given time span.

We provide a similar construct with MaxSPAN constraints. Using "Iverson brackets", a MaxSPAN constraint can be formally denoted as:

$$\phi = [\psi]_t^n \tag{6.3}$$

where ψ is the *inner constraint* that is evaluated. For ϕ to be true, the inner constraint must have evaluated to true at least n times in the last t time steps. This can be recursively defined as follows where $s_i \rightarrow s_{i-1}$ denotes that state s_i followed s_{i-1} directly in the system trace π. As before, S denotes the set of states of π. The trace must contain at least t steps.

$$s_i, s_{i-1} \in S; s_i \rightarrow s_{i-1} : [\psi(s_i)]_t^n = \begin{cases} \text{true}, & \text{if } n = 0, \\ \text{false}, & \text{if } n \neq 0 \wedge t = 0, \\ (\psi(s_i) \wedge [\psi(s_{i-1})]_{t-1}^{n-1}) & \\ \vee\ (\neg\psi(s_i) \wedge [\psi(s_{i-1})]_{t-1}^{n}), & \text{otherwise.} \end{cases} \tag{6.4}$$

Thus, in each time step—i.e., every time a state change occurs—$\psi(s_{now})$ is computed and the result saved in a history. The sampling frequency of the constraint is therefore defined by the frequency of state changes, an important difference to similar definitions such as the one by Quinton and Ernst (2012) mentioned above or the one used by Bai et al. (2012) that both use actual time spans—e.g., evaluating the number of weak constraint violations within 200ms. While it is possible that the inner constraint ψ is part of the invariant, such an arrangement might be problematic, since a violation of ψ would trigger a reconfiguration directly, whereas ϕ must not necessarily be violated. In case an escalating reconfiguration is used, however, it might be useful to reuse ψ.

The data structure to collect the historical data is a *ring buffer*. It has a fixed size and whenever an element is added, the oldest element is dropped. The position at which new elements have to be added is denoted by a *tail* pointer, the element that has to be deleted is denoted by the *head* pointer. Formally, a variable rb_t is used to denote a ring buffer with size t. Whenever ψ has been evaluated, the value $\beta = \psi(s_{now})$ is added to rb_t by pushing β on rb_t, denoted by $\beta \blacktriangleright rb_t$. As long as less than t elements are contained in the ring buffer, a new element can be added and the tail is incremented by one. If t values are contained in the ring buffer, the element marked as the head is removed, the head is moved to the next entry, and the new element is added at the position marked by the tail. The tail is then moved to the next entry as well. Figure 6.4 shows an example of a ring buffer with corresponding head and tail pointers.

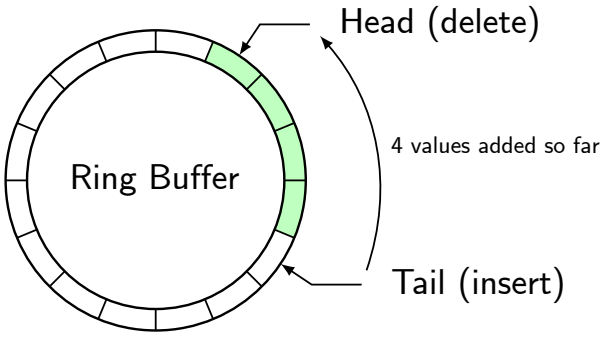


Figure 6.4: A ring buffer of size 16 with four values stored in it. The tail points to the position where the next element will be added, the head to the element that has to be removed in case the buffer is full. As the buffer is not at capacity at the moment, inserting a value will only move the tail. In case the buffer is full, tail and head will point to the same cell. The new value is stored at the designated cell and both pointers move clockwise to the next cell.

In the extended Observer/Controller model, depicted in Figure 6.3, the `MaxSpanConstraint` is a specialisation of `HardConstraint`. It contains a value for n, the maximum `numberOfViolations` of ψ, and for t, the `timespan` that is observed. The inner constraint ψ is mapped to an association to an arbitrary `Constraint`. The `RingBuffer` class has an operation `push(value)` to add a new element to the buffer and an operation `count(value)` returning the number of times `value` is present in the buffer. The class is instantiated for each instance of `MaxSpanConstraint` with the appropriate buffer size. To ensure specification consistency, $n < t$ is enforced. This is captured in the specification of the generic Observer/Controller model with the `MaxSpanConsistency` constraint:

```
context MaxSpanConstraint inv MaxSpanConsistency:
  self.numberOfViolations < self.timespan
```

6.3 System Reactions caused by Detection of Constraint Violations

Each observer maintains a list of controllers registered for specific constraints. Whenever the constraint is violated, the controller is notified and allowed to adapt the system, as shown in Figure 6.2. While system stabilisation and adaptation are described exhaustively in Chapter 8 and Chapter 9, it is worthwhile to detail the different kinds of reactions the controllers have at their disposal in case of a violation of constraints.

In the simplest case, the violation of a constraint triggers exactly one controller. This controller puts the agent into the quiescent state, thus ensuring that it performs no further actions that impede the

reconfiguration process. The controller then runs the algorithm that reconfigures the agent, possibly coordinating with controllers for other agents. An algorithm such as SPADA (cf. Section 4.1), e.g., requires the controllers of several agents to work together to find a new structure for the system. Controllers are thus allowed to interact with arbitrary other controllers and, importantly, put agents into the quiescent state without a constraint violation. As soon as the reconfiguration process terminates, the controllers send their respective agents the signal to proceed.

This example illustrates how a local constraint violation can cause a self-organisation process that changes the system structure. Thus, there is a connection between behaviour of the individual system and the behaviour of the entire system. It is often desirable, however, to limit the effects of local constraint violations and prevent their propagation through the system. In such a case, compartmentalisation techniques—such as the hierarchical structures discussed in the context of constraint monitoring in Section 6.5—can limit the effects of adaptation process to sub-systems.

Parameter Adaptation

The violation of soft constraints indicates that the system may be approaching the corridor of correct behaviour and that a violation of a hard constraint is imminent. Especially if a soft constraint is used as the inner constraint of a `MaxSpanConstraint` or as the one of the weighted constraint for a `MaxPenaltyConstraint`, a reaction to such a violation can help stabilise the system before a more drastic measure has to be taken. In such cases, controllers that try to make non-intrusive changes to the agents—such as changes to parameters that influence agent behaviour—can be used to try to avoid violating a hard constraint[1].

Indeed, violation of a soft constraint is not necessarily an indicator of an imminent violation of the corridor. That is only the case if the soft constraint has a relation to a hard constraint, either by being defined on the same parameter or by being defined on a parameter that influences those parameters that are the subject of hard constraints. In the autonomous power management example, the network frequency is observed by a soft constraint (cf. Section 7.2). If the network frequency deviates from its optimum, the balance between production and consumption is not maintained. The parameter network frequency and the production are thus coupled. A controller reacts to the violation of this soft constraint by adapting the power output accordingly (cf. Section 8.4).

"Silent" Reconfigurations

On the other hand, when soft constraints are specified that define optimality criteria, their violation can trigger a process of on-the-fly self-optimisations. The controller that is triggered by the violation can reconfigure the system in the background, looking for a new system structure or parameter set that optimises the system and fulfils the violated soft constraint again. In comparison to a "regular" reconfiguration, the controller does not transition the agent into the quiescent state in such a case. Instead, it runs a process in the background that does not affect the running system. When the process terminates because no better solution was found, the controller does not change the system and no side-effects of the silent reconfiguration become evident in the system. If, however, a new configuration was determined, the controller checks whether this configuration is compatible with the current state of the system and indeed optimises it. In this case, the controller can send the agent into the quiescent state, enact the necessary changes, and allow the agent to proceed.

Escalating System Reactions

In principle, the observer model as introduced here allows different controllers to react to the same constraint violation. This possibility can be used, e.g., to escalate the system reaction, starting from a relatively non-intrusive reaction and going to a more in-depth reconfiguration in case of a repeated violation. It can also be helpful if different algorithms are concurrently executed for the same reconfiguration task. The controllers involved in such a reconfiguration can then either select the best result or the result that has been computed first.

An escalating system reaction can be helpful if a mere parameter adaptation or a silent reconfiguration failed in the attempt to prevent the system from reaching the boundaries of the behavioural

[1] In the terminology introduced in Chapter 8, such a reaction would constitute *negative feedback* that holds the system in its current, stable state.

corridor. Section 6.1 gave a simple example for such escalating constraints: $x < 10$ was a soft constraints to which a controller could react that tries to alter a system parameter that changes the system behaviour in a way that restores x to a value above 10. A second soft constraint $x < 12$ could be used to start a silent reconfiguration in the background, allowing changes to the system structure, e.g., if that would enable x's value to drop accordingly. In case the hard constraint $x < 15$ is violated, the corridor of correct behaviour is left, the system is transitioned to the quiescent state, and a reconfiguration takes place that involves other agents in the same sub-system. The escalation could even be taken one step further by adding another hard constraint $[x < 15]^3_{15}$ that limits the number of violations of the hard constraint $x < 15$ to a maximum of 3 within 15 time steps. A violation of this MaxSPAN constraint could enforce an even more dramatic adaptation process, possibly involving sub-systems beyond the scope of the previous reconfiguration.

6.4 Black-Box/White-Box Monitoring

The generic Observer/Controller models regarded so far only consider systems in which the designer controls the architecture and implementation of the agents and has full control over their behaviour. Thus, agents always behave benevolently and disclose the required information. It can also be assumed that the information is correct to the best of the agents knowledge. If we consider open heterogeneous systems—i.e., those systems this thesis is concerned with—in which agents can enter and leave the system at will, are not under control by the designer and can behave arbitrarily, matters of trustworthiness come into play. Monitoring in such an environment is a much more challenging task. Agents might not disclose truthful information, either intentionally to exploit the system or unintentionally due to errors or other circumstances (Anders et al., 2013a). Thus, information provided by these agents about their internal state has to be taken with a grain of salt.

On the other hand, these agents might not reveal any information at all. If that is the case, the internal state must be approximated by the externally observable behaviour. This becomes especially difficult as such information can usually only be determined from the interactions with others and the environment. Instead of monitoring single agents, groups of agents now have to be observed. Such a situation can also occur in the adaptive MAS we considered so far: depending on the constraint, different views of the system might have to be considered. A regional view of the system could be helpful where an observer can monitor several agents and obtain a picture of a compartment of the system. This is due to the fact that, e.g., emergent behaviour only arises through interaction of different agents. To constrain certain behaviour, the collective behaviour of groups of agents has to be monitored. In distributed open systems, such information is, however, not readily available locally. Aggregation of the required data is therefore one open issue (Bai et al., 2012). These challenges constitute current limitations of our approach. The former challenge can be dealt with by the introduction of *black box/white box monitoring*. To deal with the latter challenge, we propose to exploit *hierarchies*, detailed and exemplified in the context of the autonomous power management system in the next section.

As we are interested in detecting changes in the internal states of agents and map these to the free variables used in constraints that define correct system behaviour, it is necessary to gather all information and the changes in this information concerning these variables during runtime. In such a case, the monitor is integrated into the internal feedback-loop of the agents which enables adaptivity and self-organisation. The O/C architecture and the mechanism introduced above ensure this by using a publish-subscribe mechanism that informs the observers whenever a state change occurs. This is an example of *white-box monitoring* in which the monitor and the observed object are coupled very closely.

In general, it is desirable to only monitor externally observable properties, but this limits the applicability of any monitoring approach severely. In case of a selfish agent, who is trying to maximize its private benefit, its information is not reliable. In addition, it might have no interest adhering to constraints that are only useful for other agents of the system but not for him. Consequently, to avoid an external interference by a controller it could tamper the information about its state. Therefore, the state of an agent has to be derived by observation of the behaviour to gain reliable information and must be considered a *black-box*. The two types of monitoring can thus be defined as follows:

White-Box Monitoring: Agents willingly disclose information about their internal state.

Black-Box Monitoring: Information about the internal state of an agent has to be inferred from externally observable properties or interactions.

In both cases, the credibility of the information is uncertain. This is clear for black-box monitoring, where biases and inaccuracies in the observation can impede the quality of the data (epistemic uncer-

tainty). But even in white-box monitoring, the information can be incorrect, either due to the fact that the agent can lie about its internal state, because the information is based on uncertain data (these are again instances of epistemic uncertainty), or that the data source is itself a stochastic process (aleatoric uncertainty). In cases of epistemic uncertainty where there is a way to check whether the information was correct or there is a second source for this information, a trust model can be used to determine its credibility and correct for systematic deviations (Anders et al., 2013a).

The realization of white-box monitoring is straightforward, as outlined in the previous sections, because all needed information is provided by the agent. All constraints can consequently be evaluated on this provided information. The evaluation of constraints in the black-box monitoring concept is much more complex. As the information about the current state of the observed agent is not directly available, it has to be inferred from observable properties.

In cases in which data collected by the observer is re-used in decision making processes, it is necessary to model the agents behaviour. Based on this model the monitor is able to state whether or not the current state fulfils its constraints and to predict how control decisions will affect the agents ability to stay within the corridor. In the scheduling of power plants, e.g., it is possible to derive the control model of the power plants from their behaviour, including minimal and maximal output as well as rate of change. Approaches from machine learning—such as support vector machines—have successfully been applied to approximating the controllability of power plants (Bremer et al., 2011). Alternatively, techniques to approximate stochastic processes such as the one in (Anders et al., 2013a) allow the formulation of such complex models from the observed behaviour. Future work might also consider the combination of these techniques to allow the machine learning approach to deal with uncertain data.

In a real system could, however, the severity of the outlined challenges might be somewhat less pronounced than in theory. As mentioned in this thesis before, an open heterogeneous system still operates under the assumption that interfaces and standards exist that the agents use to interact. These standards can prescribe access to information that is, e.g., required to evaluate constraints. Agents that try to participate and not adhere to these standards can be detected and excluded from the system. Therefore, black-box monitoring mainly applies to legacy systems or systems that have to be included although they do not adhere to the standards. If such agents exist, they pose a special case and some knowledge about their functionality should exist that can be exploited in the creation of a black-box observer.

6.5 Behaviour Monitoring in Hierarchical System Structures

If interactions between agents have to be observed or groups of agents become the focus of attention, the Observer/Controller model as introduced in the previous sections is not sufficient to evaluate the respective constraints. The model as it has been described so far only allows constraints defined on the states of individual agents, i.e., on information that can be provided by a single agent. To overcome this limitation, we use the hierarchical structure of large-scale open self-organising systems and observe constraints within individual parts of the hierarchy.

Hierarchies ensure the scalability of the system by providing information about the different parts of the system at different aggregation levels. Thus, state models can be aggregated on a higher level and monitoring can use this aggregated information to observe the system behaviour. Figure 6.5 shows the integration of the O/C concept into a hierarchical system. The principle is that an agent aggregates information about its children. Therefore, behaviours within a group of agents can be monitored by collecting information from one agent, namely the parent node of the group of agents that is of interest.

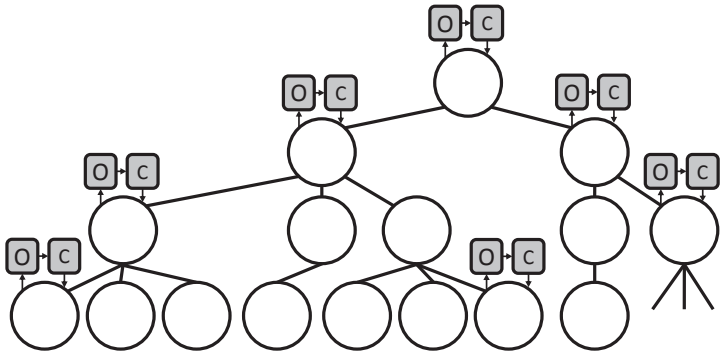


Figure 6.5: Hierarchical system structure with multiple Observer/-Controllers on different levels of the system. While each node can have an O/C, those agents controlling others have a broader view as they can request state models from superordinate agents as well. This allows them to use regional data in the adaptation process, potentially leading to quality improvements.

Observer Model for Hierarchical Behaviour Monitoring

The observer model introduced in the previous sections and depicted in Figure 6.3 can be adapted for hierarchical systems in different ways. First of all, the model does not limit the number of observed agents an observer receives state updates from. From a structural point of view, it would thus already be possible to use the model for a centralised as well as a fully distributed architecture (cf. Section 3.3). The behaviour of the observer, however, especially of the `refresh()` operation, has to be redefined for the centralised and the hierarchical case. The version depicted in Figure 6.2 uses the state model received from the observed agent directly to check the constraints. A hierarchical version of `refresh()` would have to combine the updated state model from one observed agent with the current state models of the other observed agents and use this new, combined model to evaluate the constraints.

Structurally, the hierarchical case also demands that groups of agents are observed and controlled together and yet, each of the agents has its own O/C. This introduces the requirement that not only the agents are hierarchically structured but the O/Cs as well. Information collected by the observers must flow up in the hierarchy, commands from the controllers must flow down. Therefore, the O/C model is extended as depicted in Figure 6.6. The `Observer` class is specialised to a `HierarchicalObserver` which differs only by redefining the `refresh()` operation and by introducing new associations to a superior `HierarchicalObserver` and an arbitrary number of subordinate `HierarchicalObserver`s. The consistency of the hierarchy is ensured by an OCL constraint that ensures that superordinate observers have the correct instance of `HierarchicalObserver` as their superior[2]. All observers in a hierarchical setting are instances of `HierarchicalObserver`.

```
context HierarchicalObserver inv HierarchyConsistency:
self.subordinateObserver -> forAll(ho:HierarchicalObserver | ho.superiorObserver = self)
```

The `refresh()` operation is redefined by first calling its superiors `refresh()` with the new state model before performing the same steps depicted in Figure 6.2. This ensures that the superior is always informed about changes in the subordinate agents' status. To prevent information from spreading to far upwards in the hierarchy, a `HierarchicalObserver` only informs its superior if an observed agent is the source of the call to `refresh()`.

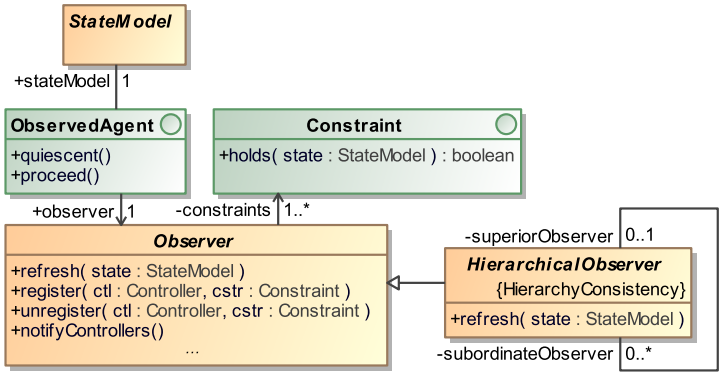


Figure 6.6: The Observer/Controller model and its extension with a `HierarchicalObserver`. The new abstract class can be used instead of the standard `Observer` class and adds associations to the observer on the hierarchy level above and to the observers on the hierarchy levels below. In addition, an OCL constraint to check the consistency of the hierarchy is supplied. The `refresh()` operation is redefined as well to accommodate information propagation.

Every time a state model update occurs, the HierarchicalObserver has to combine—or *synthesise*—the current models of the agents into a combined state model. This combined model is then used as a parameter for calls to the `holds()` operations of the constraint known to the observer. It is thus possible to define constraints over the attributes of this combined model.

If a higher-level O/C monitors many underlying agents, however, the computational cost and messaging involved in maintaining this regional view of the system can become very large. In such cases, techniques to *abstract* state models can be helpful. We believe that an approach very similar to the abstraction of solution models as presented in Chapter 11 can be employed for this purpose. Further analysis of this proposition is however, left as future work.

[2]Please note that the UML associations used to denote the hierarchical structure of observers will not translate into fields of the `HierarchicalObserver` class in an actual distributed system. The transformation of the model into code must, instead, ensure that messages can be sent between the `HierarchicalObserver`s and that the appropriate communication endpoints are known.

Coordination of O/Cs in Hierarchical Systems

The systems of systems formed in a hierarchical structure such as the one depicted in Figure 6.5 provide boundaries for information flows and compartmentalise the system. Lower-level structures are unknown to high-level structures. Their behaviour is subsumed by their superior agent. Coordination between agents and between their observer/controllers is thus usually limited to the superior/subordinate relation. The superior can make decisions about the subordinates who provide it with the necessary information. Such a strict information flow is used, e.g., in HiSPADA (cf. Chapter 4) and for power plant scheduling (cf. Chapter 9). Figure 6.7 illustrates this information flow.

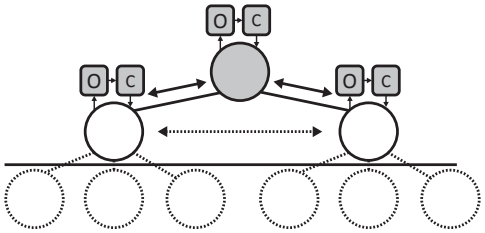


Figure 6.7: The agent indicated in gray directly controls two other agents. The black solid bar denotes its scope. Information usually flows as indicated by the solid black arrows, with state models being propagated to the superior and commands being sent to the subordinates. However, in some cases (cf. the SPADA algorithm described in Section 4.1), direct communication between the O/Cs on the same layer can be useful. The superior will then act as an intermediary to establish communication between its subordinates.

This limitation ensures scalable operation since information and decisions are regionalised and do not distribute through the entire system. It implies that decision problems are smaller since they only affect a smaller number of agents and that information aggregation works on smaller data samples as well. At the same time, since no global information is available, all solution approaches are necessarily heuristics, only achieving a best-effort performance as compared to the possibly optimal solutions a centralised decision procedure with global knowledge could find.

For some algorithms, a strictly vertical flow of information and control is too confining. In the case of the SPADA algorithm described in Section 4.1, Observer/Controllers on the same level of the hierarchy interact to create a partitioning of the system—i.e., a division into sub-systems—that fits certain predefined quality criteria. A horizontal information and control flow thus has to be established. The superior agent can act as an *intermediary* in such cases. As soon as one of the agents it controls detects that a reorganisation is necessary because constraints have been violated, it can request information about its siblings from its superior. The superior can then communicate the communication end points of its subordinates to the requester, enabling the observer/controllers to interact with each other. After the reorganisation has been completed, the superior now controls a different set of agents. This principle is used in the reorganisation of a hierarchy level with HiSPADA (cf. Chapter 4). Please note that a level of compartmentalisation is still maintained since the reorganisation is limited to the children of the superior agent.

If the hierarchy is flat, i.e., an agent from the lowest level assumes the role of the intermediary and hierarchy levels are not nested, it is not necessary to use `HierarchicalObserver`s. Instead, the standard observer model can be employed and information and control flows horizontally between the O/Cs. The standard implementation of SPADA, e.g., uses such a simplified hierarchical model.

Hierarchical Behaviour Monitoring in the Autonomous Power Management Case Study

In the case study, AVPPs act as intermediaries and observe and control the underlying agents. They are explicitly represented by agents and are equipped with several O/Cs for the requirements outlined in Chapter 7. Power plants have O/Cs as well for specific purposes. The overall system architecture resembles the hierarchical observer/controllers depicted in Figure 3.2c introduced in Section 3.3, with the important difference that the SuOC is always explicitly represented by an agent.

Observation and control for each individual power plant The main functionality of the individual power plants is to participate in the creation of schedules by, e.g., creating output predictions, and to adhere to these schedules. For this purpose, the agents control a physical power plant interface. The O/Cs of these agents observe the network frequency and adapt their output accordingly to keep it close to 50Hz (cf. Section 7.2 and Section 8.4). They can also observe the

physical interface of the power plants and, e.g., detect problems in the turbines or other essential hardware. This information can be propagated to the controlling AVPP so that appropriate measures can be taken.

Observation and control for AVPPs An AVPP's main task is to create the schedules for the individual power plants to satisfy a given demand (cf. Section 9.1). It collects information from the controlled power plants and uses one of several solution procedures to create output levels for controllable plants based on predictions of output of stochastic power plants and of the demand. Its adaptive behaviour is mainly related to the structure of the system, i.e., which power plants it controls and which AVPP it is controlled by. Thus, the AVPP observes the difference in aggregate credibility and compares it to its neighbours. It also monitors the time required to create the schedules. Whenever these constraints are violated, appropriate controllers reorganise the structure, e.g., to ensure that all AVPPs have a similar mixture of credible power plants and that scheduling times stay within certain bounds (cf. Section 4.4, Section 7.3, and Section 9.1).

As a hierarchy has already been established on the agents, it is not necessary to establish an additional hierarchy on the O/Cs. As indicated in Figure 6.7, vertical information flow is usually sufficient in this scenario. An an intermediary, the AVPP also establishes the black-box, since information only flows between layers (i.e., from AVPP to power plant or from controlling AVPP to subordinate AVPP—this is possible because an AVPP itself behaves like a controllable power plant to the outside).

An alternative to introducing explicit representatives of systems of systems, one of the constituting agents can take on the responsibilities of the intermediary. This is not practical in the given scenario since an agent representing a power plant serves a particular legal entity and thus does not have incentives to make impartial decisions.

6.6 Monitoring and Observation in the Literature

Some of the related work on monitoring and observation has already been cited in this chapter. This section summarises the work performed elsewhere and first details the theoretical foundations before outlining concrete algorithms and frameworks.

General Approaches and Theoretical Foundations

The observation of systems, especially adaptive ones, has received much attention in recent years. The formal methods community has, e.g., coined the term *runtime verification* to denote approaches in which systems are no longer verified at design time by checking a specification against the functional requirements, but in which the system behaviour is compared to the specification at runtime. This step is motivated by the failure of classical verification methods such as theorem proving, model checking, and—to a lesser extent testing—to deal with the complexity and dynamics of modern systems (Leucker and Schallhart, 2009). In the spirit of Organic Computing, the verification task is thus delegated to runtime. The techniques introduced by Leucker and Schallhart (2009) bear a similarity to what is described here since they use constraints for the specification of correct behaviour as well. However, there is no correcting reaction to a constraint violation. If a constraint is violated in runtime verification, a proof that the system actually does not adhere to its specification is found. In the systems we are concerned with, we include this detection in a feedback loop constituted by Observer/Controllers (see Chapter 8) that allows us to react to a violation and return the system into a correct state. The notion of correctness is thus slightly different in our case, implying a sort of "best-effort" correctness.

A quantitative approach to runtime verification of self-adaptive systems is presented by Calinescu et al. (2012). A probabilistic model checker is used to check a global model of the system state against requirements defined in temporal logic. Similar to Leucker and Schallhart (2009), a three-valued logic (including an "undecided" value) is used. The monitor is part of a MAPE-cycle (cf. Section 3.3) and thus allows a reaction to the violation of the formal requirements. The authors propose to use the model checker's output in the planning phase to determine the best course of action for redirecting the system to exhibit correct behaviour. As with RIA, the system can thus leave the corridor of correct behaviour until a reconfiguration has taken place. The main drawback of the approach is the necessity to define and maintain a consistent global state model. Combining distributed information at runtime consumes time and ensuring that the model actually represents the current state of the system is difficult. The authors thus see the development of techniques to ensure this requirement as one of the most important lines of future work. Even if appropriate methods could be found, however, the

complexity of maintaining and checking the model is very high. As historic data has to be carried along, the memory consumption increases with each time step. The model always has to be checked in its entirety, even if only small, local changes occurred. In comparison, the method proposed in this thesis compartmentalises and localises the effort to create state models and checks on them are very simple.

A very general study on monitoring was conducted by Goodloe and Pike (2010) for safety-critical real-time systems. They stipulate a number of requirements for monitors in distributed systems, including that the monitor should be a separate process, running in parallel with the process being monitored. Such a separation can be achieved by instantiating the observers as independent agents that communicate with the monitored agents with messages. In accordance to our own views and those of Calinescu et al., the authors also emphasise that after a violation of the system constraints, the system should be restored to a correct state. Goodloe and Pike also outline a number of different architectures for the monitor that differ in the "view" of the system. The view defines whether a singular component, groups of components, or the entire system is observed by one monitor. The local architecture—in which one component is observed by one monitor—is very similar to the hierarchical observer/controller architecture introduced in Section 3.3 and applied in Section 6.5.

Monitoring Algorithms and Frameworks

Monitoring individual system parameters is in the focus of several authors, e.g., Jin et al. and Roşu. They usually provide specialised algorithms that allow checking states against *Past Time Linear Temporal Logic* (ptLTL) (Manna and Pnueli, 1992, 1995) terms and thus observe the correctness of these terms at runtime. Since runtime monitoring only yields data about past and current behaviour, standard temporal logic operators such as "always" (depending on the logic used, either $\Box$ in LTL or AG in CTL) can not be evaluated since the future behaviour of the system is unknown and the semantics define the operator on the entire (possibly infinite) system trace. This is circumvented in ptLTL by only defining operators that work on prefixes of the trace and thus only in the past and present. These can be checked by monitors that save the system trace, similar to the way MaxSPAN constraints are handled in this work. The authors also use the monitors in unit testing, a feature that we have yet to explore. The *JavaMOP* system (Jin et al., 2012) operationalises many of the monitoring algorithms presented, e.g., in (Roşu, 2007; Roşu et al., 2008). However, all mentioned related work is only used to observe singular variable values, not entire system properties. Our approach uniquely provides a method for decomposing the system properties to such constraints. In addition, the connection of the monitoring with a system reaction and thus a controller part is not provided.

The *Dresden OCL Kit*[3] (Demuth, 2004) provides tools to check UML models for consistency by evaluating OCL statements on them (Demuth and Wilke, 2009). This validation can either happen at design-time to check whether changes in the models still adhere to predefined correctness criteria. It can also happen at runtime, though, to check whether the system dynamics have violated any of the predefined constraints. Technically, aspect-oriented programming is used to instrument the byte code of the class implementations at all points where the attributes the constraints are defined on change. The rationale is that OCL constraints can only be violated if the state of the instances change and this occurs at exactly these places. While this approach can be helpful in cases where the agents are implemented by the system designer and only local observation is required, it has drawbacks in distributed development settings or in open systems in which the developers do not have access to the agent source code. The publish/subscribe approach described above provides a clear interface. Due to the byte-code instrumentation employed by Dresden OCL, it is also limited to the Java Virtual Machine and programming languages that compile byte-code for this runtime environment.

A slightly different approach is pursued by Bai et al. (2012). The algorithms developed by Roşu and Jin et al. are adapted for use in distributed, reactive systems. The main challenge is that the monitoring algorithms assume that the properties can be checked on traces of a transition system and thus imply synchronicity, an assumption that does not hold in distributed systems in which different components work asynchronously. Intuitively, a monitoring algorithm requires a consistent global state to verify properties that combine several variables that are distributed over different components. Changes on these variables thus need to be collected centrally and it has to be ensured that the overall system state is known. The wrapper ensures these requirements by collecting and aggregating all necessary information. The techniques provided here address this problem by either decomposing the constraint

[3] `http://www.dresden-ocl.org`

so that it can be checked on a single component with a decentralised observer/controller architecture or by delegating the monitoring task to a higher level in the hierarchy with a hierarchical O/C architecture.

A framework similar to JavaMOP is *Monitoring and Checking framework (MaC)* presented by Kim et al. (1999), specifically designed, however, to observe correctness properties of real-time systems in the current state. MaC allows the definition and observation of timing and real-time properties as well. For this purpose, the system is checked with a given sampling rate. In comparison to other approaches, MaC observes individual methods in the programmes that are monitored. Byte code instrumentation is used to ensure that the checks occur at the correct point and at the correct times. Again, no reaction to a violation of the system specification is considered.

Chapter Summary and Outlook

Monitoring system behaviour is the foundation of the Restore Invariant Approach. This chapter introduced the use of both hard and soft constraints for this purpose and proposed an architectural blueprint based on the Observer/Controller architecture suitable for most adaptive and self-organising systems, specifying both the structural elements as well as the communication between those elements for monitoring. These principles work well in hierarchical systems in which agents willingly disclose internal state information and is compatible with a number of relevant soft constraints formulations. The monitoring framework suggested here is the foundation of a semi-automatic observer synthesis process detailed in Chapter 7 that allows the generation of design artefacts and code for monitoring from the requirements documents.

Chapter 7

Synthesis of Observers from Requirements Specifications

Summary. The constraints describing the correct behaviour of the system and thus its specification are a product of the requirements engineering activities that are performed during the early phases of a software engineering process. Pre-empting the more detailed description in Part V, this chapter introduces the technique with which the requirements models are annotated with formal constraints and the transformation process that yields the Observer/Controller instantiation for the system under construction. It discusses which constraints can be covered by the automatic model transformation, which adaptation and extension points exist, and which manual steps a developer has to perform in order to adapt the technique to a specific target system.

Publication. A summary of the concepts and techniques outlined in this chapter has been published in (Steghöfer et al., 2013c). A more detailed description of the approach with an additional example can be found in (Eberhardinger et al., 2013).

The generic Observer/Controller model presented in the previous chapter has to be instantiated for a specific system to observe the constraints of that system and react accordingly. Additionally, the monitoring infrastructure has to be tailored for each system if special forms of communication are used or other specialisations of the generic architecture are necessary. Instead of performing such adjustments manually for each new system, a model transformation process can be used in which the generic infrastructure, the specific system model, and the constraints that have to be observed are used to automatically create large parts of the observation infrastructure. This synthesis process ties into an agent-oriented software engineering approach that uses incremental development to refine a system architecture and the system's code in each iteration, such as the one described in Part V. In fact, the constraints of the system as well as the specific system model on which the transformation are executed can be gained during the requirements elicitation phase as outlined in Section 7.3.

If models are used extensively during software development, the term "model-driven design" (MDD) is often used. However, since almost all modern software design processes rely heavily on models in all phases of the process, this term does not bear much differentiating meaning any more. The transformation process described in the following adheres to the principles of the *model-driven architecture* (MDA), a software development framework championed by the Object Management Group (OMG)[1]. The idea behind MDA is to use models during the entire life cycle of software development that are successively refined by transformations. For this purpose, the OMG has created a number of specifications, including UML and the Meta Object Facility (MOF) that forms the basis for many of the other languages and meta-models. The driving philosophy behind the model-driven architecture is reuse of models. In principle, the designer starts out by creating computation-independent models (CIMs), containing information about the requirements and the context of the system without any reference to its design or structure. This abstract model is then refined—by a set of model transformation—to a platform-independent model (PIM) that contains computational details but does not make any as-

[1] A good starting point for information about MDA is (Truyen, 2006). If not specifically identified, the information in the following is a summary of this introductory text.

sumption about the implementation platform (or platforms) to be used. Finally, another set of model transformations is used to create one or more platform-specific models that augment the PIMs with specific information about the implementation platform and environment.

More specific models contain increasingly more information about the execution environment and the implementation details. Thus, they get closer to the code that will finally be produced. This information can contain references to services provided by the platform, constraints that have to hold on specific platforms, or assumptions that can be made. Ideally, the final code is also created by a code transformation. If different target platforms are used within a system, transformation for each of them should be provided. This puts the definition of the actual code transformation into the focus of the development team. Instead of creating code from the models, they create transformations that—ideally—create the code for them. The models act at the same time as the transformation source, as documentation of the system, and as communication tools within the development team as well as with external stakeholders and the customer. The use of these intuitive, domain-specific models thus increases productivity and quality (Sendall and Kozaczynski, 2003).

This chapter introduces a process that adheres to the MDA's terms of reference and successively refines a computation-independent model, provided as a requirements and a domain model, into a platform-independent observer model and then into platform-specific implementation code. It can be fully embedded in an agent-oriented software engineering process as detailed in Chapter 13. Section 7.1 gives an overview of the transformation process. The creation of the CIM is explained in Section 7.2. The CIM-to-PIM-transformation is then detailed in Section 7.4. Finally, Section 7.5 details the transformation to platform-specific code.

7.1 The Transformation Process—From Requirements to Observers

The synthesis of the monitoring infrastructure follows a model-driven process, outlined in Figure 7.1, and is divided into three major steps which can easily be integrated into an iterative-incremental design process. The process can be repeated when requirements or the domain model change in a model-driven design (MDD) approach. Changed parts of system models and implementation will be re-generated while existing models and code are preserved.

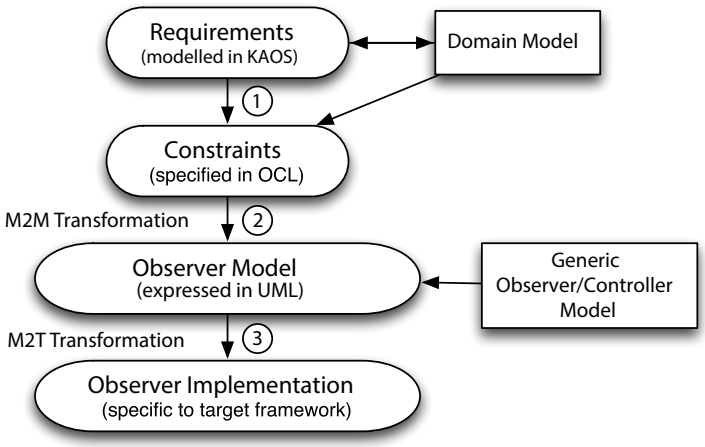


Figure 7.1: The model transformation process, starting from requirements modelled in the KAOS methodology (cf. Section 7.2) that are formally described while the requirements become clear and a domain model is elaborated, to abstract observers expressed as UML class and activity diagrams backed by a model of the Observer/Controller (cf. Section 7.4), to the final implemented observers for the target platform (cf. Section 7.5).

Step 1: System goals and constraints are described formally during the iterative process of requirements analysis, shown in Figure 7.2. During this process, a domain model is created that can be used to express constraints in OCL (Object Constraint Language)—a language arguably more accessible to system designers than the aforementioned temporal logics—that formally describe the requirements (see Section 7.2). These OCL-constraints $\phi \in \Phi$ define the correct states of the system, as shown in Eq. (6.1), and are therefore transformed in a model-to-model transformation to abstract observers (see Section 7.3) when moving from the analysis state to the design stage in each iteration. The state-based evaluation of the OCL-constraints, i.e., to check for all $\phi \in \Phi$ whether $\phi(\sigma_i)$ with $\sigma_i \in S$ holds, conforms well with the semantics of the transition system RIA is based on as described above.

Step 2: The underlying structure of the observer model, which is created during the transformation process described in Section 7.3, contains the refined elements from the domain model and the

generic Observer/Controller model. In addition, for each agent that has to be monitored, an observer class is created and for each OCL-constraint, a new abstract constraint class is created. Furthermore, the validity check for each constraint $\phi(\sigma_i)$ is modelled as a UML activity diagram in the constraint classes.

Step 3: The observer model is transformed into code from which the actual distributed observers for the agent system can be compiled (see Section 7.5) when moving from the design to the implementation stage. This last transformation is highly specific to the target system.

If the generic observation model proposed here is used, the software engineer has four responsibilities left: (1) formally describe the constraints based on a domain model; (2) create platform-specific transformation rules for the target platform; (3) create code for the controller and (4) for the state update. The rest of the infrastructure is created by the transformations.

The transformation steps are automated as described in the individual steps below. This means that the developer only has to create the requirements model, define hard and soft constraints, and the conceptual model. Starting from these artefacts, the automatic model transformation can create the specific observation model and from that implementation source code. If required, the developer has the possibility to adapt the generic Observer/Controller model and the transformation as required. The work flow remains the same.

7.2 Requirements Modelling and Constraint Derivation for the Decentralised Power Management Case Study

The first step in the process is the creation of the computation-independent model (CIM), consisting of the requirements model and a domain model. These two models develop in parallel, as new requirements imply structures and data elements that are necessary which are captured in the domain model. The emerging model of concepts and their relation to each other can, in turn, inform new requirements or alter existing ones. This way, an iterative approach to the modelling of the requirements and the creation of the domain model is established.

To specify the requirements we use the goal-oriented requirements specification methodology KAOS, suggested by von Lamsweerde and Letier (Lamsweerde and Letier, 2004). Basically the KAOS methodology consists of a graphical and a textual, formal description of the desired system. The graphical notation describes and connects goals, requirements, agents and obstacles. The connections specify refinements, assignments, or obstructions. These elements are used to model a goal refinement graph with specific system requirements at its leafs (cf. Figure 7.3). To build this graph we are using only a subset of the methodology, i.e., the refinement consists only of logical *and*s, implying that all requirements have to be achieved to fulfil the global system goal. The requirements are modelled in Objectiver (RespectIT, 2013) and exported in Eclipse XMI format for direct integration into the model-driven process.

Use of KAOS features
KAOS offers many more features than the ones used here, including an *operation model* detailing the state transitions of objects defined in the *object model*. We use standard UML to describe these elements. KAOS also has a structured language with which goals should be defined, including maintain, achieve, and cease goals. In many cases, we use natural language to describe goals and requirements instead as this makes the diagrams easier to understand. For a detailed description of all elements of the language, please refer to (van Lamsweerde, 2001).

We use OCL instead of linear temporal logic (LTL) proposed by Dardenne et al. (1993) and Lamsweerde and Letier (2004) for the formal description of these requirements. These authors have suggested to complement the process of requirements elicitation with a process of formal specification, identifying not only constraints under which the system operates but also specifying the behaviour of operations and the semantics of events. This takes the approach we pursue one step further: instead of focusing on correct system states and defining a corridor of correct behaviour in which the system operates, the constraints and invariants in Dardenne et al.'s (1993) approach prescribe the system behaviour in complete detail, in effect yielding a specification that can be executed and subjected to formal verification. Such an approach can be helpful when developing safety-critical systems of manageable size, but becomes burdensome for larger systems. Especially maintaining the formal specifications as long as the system requirements still change is difficult and error-prone. We have argued before (Nafz et al., 2011)see, e.g., that sufficiently complex, self-organising systems can no longer be formally specified and verified this way and must thus be tackled differently. The Restore Invariant Approach and the corridor of correct behaviour are

only concerned with allowing adaptations to errors that occur at runtime. The concepts developed by Dardenne et al. and Lamsweerde and Letier are thus re-used under this aspect.

Goal-driven Requirements Engineering with Uncertainty

Cheng et al. (Cheng et al., 2009) propose an extension of the KAOS methodology that allows to express requirements for adaptive systems by incorporating uncertainty factors the system is supposed to adapt to. These uncertainties are identified with the help of the conceptual domain model. Their existence can lead to a reformulation of requirements, introduction of new requirements, or a change in existing ones, effectively introducing requirements for system adaptivity. The proposed process, illustrated in Figure 7.2 and detailed in the following, is structured into four parts, which are performed iteratively. Since it uses progressive refinements from system goals to individual requirements of the agents, it can be easily integrated in an iterative-incremental software engineering process.

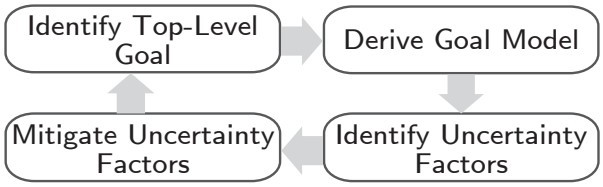


Figure 7.2: The requirements engineering process, according to (Cheng et al., 2009), starting with the identification of the top-level goals of the system, which are refined in the next step. The resulting goal model is afterwards examined according to find uncertainty factors. This could be mitigated, e.g., by introducing new goals. This process is iteratively executed until all uncertainties are eliminated.

Identify Top-Level Goal: The refinement process starts with a global goal for the intended system. Based on this global goal the top-level goals are derived which are necessary to fulfil the global goal. System goals have to be achieved through appropriate agent and system behaviour. In large-scale open self-organising system, this is often achieved with emergent behaviour.

Derive the Goal Model: The top-level goals are the basis for the complete goal model, which is developed by refining the goals until clear and achievable requirements can be formulated for the fulfilment of the goals. A requirement is a goal that is assigned to an agent. This relation assigns the functionality specified in the requirement to the agent, indicating that the implementation of the agent has to fulfil the requirement.

Identify Uncertainty Factors: This goal model has to be examined to identify uncertainty factors, i.e., obstacles that might prevent the system from reaching the goals. These obstacles are assigned to the goals they hinder. They can be refined in the same fashion as goals.

Mitigate Uncertainty Factors: Obstacles can be mitigated in four different ways, sorted by their impact and by the way they should be applied. The mitigation process is what introduces adaptivity in the system. The first mitigation strategy is to ignore the obstacle. This can be useful in cases where the obstacle is outside the control of the system or does not impede the goals in a way that the effort of mitigation is worthwhile. The second possibility is to add new goals or requirements on a low level. These goals can again be refined. If this strategy proves insufficient, it is possible to relax the goals. Cheng et al. (2009) introduces a language—RELAX—for this purpose that allows to express relaxations like “as often as possible”. Of course, such a relaxation is only advised if the goals remain meaningful. Finally, as a last resort if relaxation is not possible, new high-level goals can be introduced, again starting a process of successive refinements.

After mitigating the uncertainties, the process has to be repeated, until a sufficient model is developed. As part of this refinement process, we propose to augment requirements with formal specification of system goals and constraints in OCL. These OCL-constraints will be monitored by the observers and are formulated on the concepts defined in the domain model.

Example: Constraint to observe the network frequency in power grids.

In autonomous power grids, scheduling of controllable power sources is performed based on predictions of output and demand. These predictions are based on a number of uncertain factors. Therefore, even the best scheduling algorithms will never be able to approximate the required demand and the so called

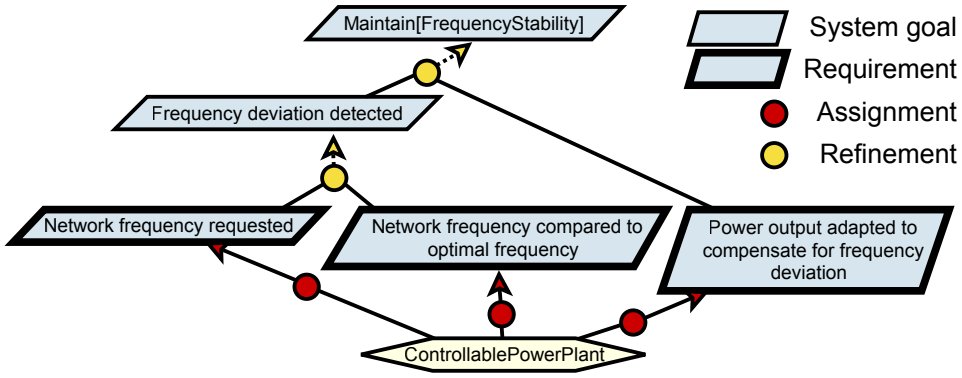


Figure 7.3: The goal refinement graph for the goal "Maintain[FrequencyStability]" in the graphical notation of KAOS. Requirements are refined from goals and assigned to agents.

"residual load", i.e., the power that needs to be produced when all production by non-controllable power plants (solar, wind, residential heat-and-power) has been factored in. However, the power grid is very sensitive to deviations between production and demand and therefore, there needs to be an adaptive mechanism that can quickly react to such deviations. Since deviations alter the power grids internal frequency, all power plants can monitor this frequency and react to deviations from the optimal frequency autonomously.

The necessity of adapting the power plants' output based on the network frequency is captured in the goal "Maintain[FrequencyStability]". It can be refined to concrete requirements for controllable power plants as shown in Figure 7.3. They need to measure the frequency and compare it to the optimal frequency, reacting to deviations by adapting their output. These requirements capture the control loop: changing the output has a direct effect on the network frequency, thus providing feedback. The constraint that needs to be observed corresponds to the requirement "Network frequency compared to optimal frequency". In OCL it can be expressed as:

```
context ControllablePowerPlant soft noFrequencyDeviations:
  (currentFrequency - optimalFrequency).abs() < allowedDeviation
```

While this constraint may seem simplistic, it is a good example for a property that has to be monitored as part of a feedback loop in an adaptive system. If the `allowedDeviation` is exceeded, the constraint is violated and a controller is informed, as outlined in Chapter 6. The controller then adapts the system to return to the corridor of correct behaviour, in this case, by changing the output of the power plant to decrease the frequency deviation.

The requirements and especially the corresponding OCL constraints inform the creation of the domain model, depicted in Figure 7.4 as a UML class diagram. As the requirements in Figure 7.3 are assigned to `ControllablePowerPlant`, an according concept has to be introduced. It is already clear at this early stage that controllable power plants will become independent agents. The OCL constraint in Listing 7.2 references the attributes `currentFrequency`, `optimalFrequency`, and `allowedDeviation` which are also included in the domain model. The other elements depicted in the figure will be introduced later on in the chapter.

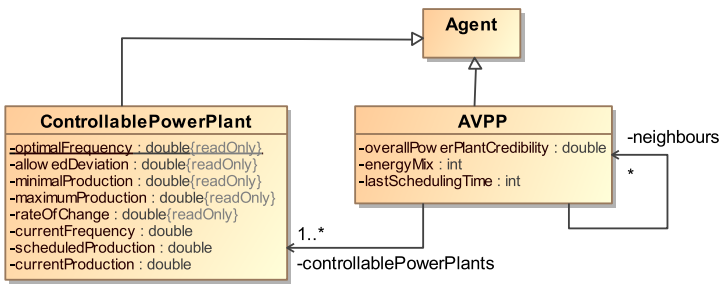


Figure 7.4: The simplified domain model for the autonomous power management case study. The elements of the domain model are informed by the requirements and the data that is required to express the constraints of the system.

Requirements Modelling for Systems of Systems

Capturing system goals and requirements for systems of systems makes it necessary to consider the compartmentalisation that characterises the running system during requirements elicitation. In particular, the system levels, boundaries, and the division into sub-systems has to be taken into account. For each of these elements, requirements can be defined separately, based on assumptions about the behaviour of other parts of the system. Such an approach is very similar to the way relies and guarantees are specified (cf. Section 7.6).

The task is simplified by the fact that KAOS enforces the assignment of requirements to individual agents. If sub-systems are represented by agents, this allows assigning requirements to sub-systems directly, thus aiding in the compartmentalisation of the system. The requirements shown in Figure 7.3, e.g., are assigned to the `ControllablePowerPlant` agent, and thus to the micro-level. Figure 7.5, on the other hand, shows requirements for AVPPs and thus for the meso-level of the system. High-level system goals are usually fulfilled by agents representing higher-level system structures or broken down into individual requirements for the micro-level agents that have to fulfil these goals in a bottom-up process (cf. Sudeikat et al., 2012).

To aid in this compartmentalisation on the requirement level, the system designer has to make sensible choices of what constitutes a sub-system. Identifying the micro-level agents is often simple as they represent physical entities or the stakeholders of the system. The iterative process of refining goals to requirements aids in the identification of sub-systems since it makes it easy to localise the fulfillment of system goals. As requirements that have to be monitored by the system at runtime are specified formally, it is necessary to identify the attributes of the combined state model required for hierarchical monitoring (cf. Section 6.5). Therefore, a good rule of thumb is "if more than local information is required to fulfil a requirement, it has to be assigned to a higher-level agent". If done correctly, this process provides "automatic" localisation of constraints both with respect to the observing agent and the required hierarchy level. Thanks to the parallel development of requirements model, domain model, and controller specification (cf. Chapter 9), errors are identified early and can be detected and fixed before an advanced design makes it difficult to change things.

Incorporating Black-Box and White-Box Monitoring

As discussed in Section 6.4, agents that do not willingly disclose information for use in the monitoring infrastructure have to be treated with special care. During the early phases of requirements elicitation, it is necessary to identify whether such agents, e.g., legacy systems, exist in the system. If they do, this has consequences on their state models and on the constraints that can be defined on their states.

In any case, all relevant data has to be part of the state models used in the monitoring infrastructure later on. In the domain model, however, these state models are not yet present. Instead, the data has to be part of the concepts identified there as explained earlier. If the domain model becomes complex, it might be possible to denote the relevant data with a UML stereotype, e.g., «observed». This stereotype can be used in the model transformation later on to identify relevant data and

7.3 Types of Constraints and Controller Specification

The example in the previous section specified a *hard constraint* that has to hold at all times in the system. Its violation must cause a reaction of the system to maintain correctness at runtime. Therefore, the definition of the behavioural corridor with hard constraints is a definition of functional correctness at design time that enforces functional correctness at runtime. As discussed previously—e.g., in Section 6.2—hard constraints can be complemented by soft constraints to further specify a section of the corridor that is considered "better" than the other parts within the corridor, thus allowing assertions about the quality of solutions.

Soft Constraints

There are three possibilities to define soft constraints during the requirements elicitation process.

Use the keyword "soft" in the constraint formalisation The constraint in Listing 7.2 has been specified as a soft constraints simply by using the soft keyword. The system designer can decide that a constraint should be soft whenever she feels that a violation should not transition the system to a quiescent state.

Use KAOS's soft goals KAOS contains the concept of *soft goals*, used "to express that subgoals are expected to achieve the parent goal within acceptable limits" (van Lamsweerde, 2001). As a rule of thumb, requirements derived from such a soft goal can be expressed as soft constraints.

Use Cheng et al.'s (2009) RELAX goals To mitigate uncertainty, and accommodate adaptivity, requirements that refine goals that have been RELAXED by adding qualifiers such as "as often as possible" can be expressed as soft constraints. However, this formulation is somewhat vague and using a MaxSPANconstraint is encouraged.

At the moment, only the first variant is supported. Each soft constraint thus must be indicated by the keyword `soft`. The transformation rules can, however, be easily extended to cover the other possibilities. Independent of the way the soft constraint is formulated, it is treated specially in the transformation process later on. At runtime, soft constraints can be violated without causing an immediate system reaction. They can also participate in constraint relationships as described below.

MaxSPAN Constraints

To define a MaxSPAN constraint during requirements analysis, the **`maxspan`** stereotype can be used. It is an addition to standard OCL syntax and takes two "arguments", the allowed maximum number of violations (n w.r.t. Equation 6.4) and the timespan (i.e., t). The following constraint uses the keyword to define a constraint limiting the change in production to half of the possible change in 7 of 10 time steps to ensure that the power plant is not constantly powered up and down:

```
context ControllablePowerPlant maxspan(3,10) rateOfChangeLimitation:
 (self.scheduledProduction - self.currentProduction).abs() > self.rateOfChange * 0.5
```

MaxPenalty Constraints

MaxPenalty constraints use a similar syntax. A **`maxpenalty`** stereotype is used and a single argument is required—the maximum penalty that is tolerated by the constraint. To specify which constraints are regarded in the calculation of the penalty, a set of constraints is used. This is, again, non-standard OCL syntax but similar to how sets are defined in OCL. The following constraint restricts the maximum penalty for the constraints `c1`, `c2`, and `c3` to 34:

```
context ControllablePowerPlant maxpenalty(32) c4:
 {c1, c2, c3}
```

In order to use weighted soft constraints, the constraint relationship that can be transformed into those weights have to be defined. For this purpose, the developer creates a separate model. A metamodel and a respective editor are available. Since all constraints are identifiable by name, constraint relationships can easily be written as $c_1 \gg c_3$ and $c_2 \gg c_3$ and so on. The procedure introduced in Section 5.4 can be used to calculate the weights of the individual constraints. In addition, a validation step occurs which checks that all constraints that are related to each other this way are observed by the same `Observer`. This is necessary to ensure that the `MaxPenaltyConstraint` can be evaluated. Soft constraints that are part of a constraint relationship are automatically transformed into a `WeightedSoftConstraint`.

Physical Constraints and Controller Specification

Not all identified constraints will have to be observed at runtime, however. Especially if the system includes components that interface hardware, requirements can be translated into constraints that express the physical restrictions of the hardware. In the autonomous power management system, e.g., the power plants have requirements that express that they have to be operated within their physical restrictions. Intuitively, this implies that the system has to ensure that the minimal and maximal output of the power plant is considered. As the power plant is not able to operate beyond these limits, however, it is nonsensical to observe these constraints and react to their violation at runtime. Instead, the scheduling of power plants, as presented in Chapter 9 has to take these constraints into account.

Therefore, physical constraints are distinguished from invariants and soft constraints by using the keyword **`phy`**. The physical constraints on power plants thus translate to:

```
context ControllablePowerPlant phy maximalProduction:
  currentProduction <= maximalProduction
```

```
context ControllablePowerPlant phy minimalProduction:
  currentProduction >= minimalProduction
```

Of course, the class `ControllablePowerPlant` needs to have an attribute `currentProduction` and constants `maximalProduction` and `minimalProduction`, respectively. These attributes have to be part of the domain model and of the final design documents later on. While it is natural to define these constraints for the controllable power plants directly, they are indeed used in a scheduling process, as outlined in Chapter 9. The controller that has to adhere to these constraints is thus part of the controller of the Autonomous Virtual Power Plant (AVPP) that performs the scheduling. As this is the case, adhering the physical limitations of the power plants becomes a responsibility of the AVPP. For this reason, the requirements model as depicted in Figure 6.7 assigns these constraints to the AVPP instead of the power plants. Thus, they have to be redefined to work in the context of the AVPP and to reflect their influence on the scheduled production, i.e., the power they have to produce in the next time step[2]:

```
context AVPP phy maximalProduction:
  self.controllablePowerPlants->forAll(c: ControllablePowerPlant |
                                        c.scheduledProduction <= c.maximalProduction)
```

```
context AVPP phy minimalProduction:
  self.controllablePowerPlants->forAll(c: ControllablePowerPlant |
                                        c.scheduledProduction >= c.minimalProduction)
```

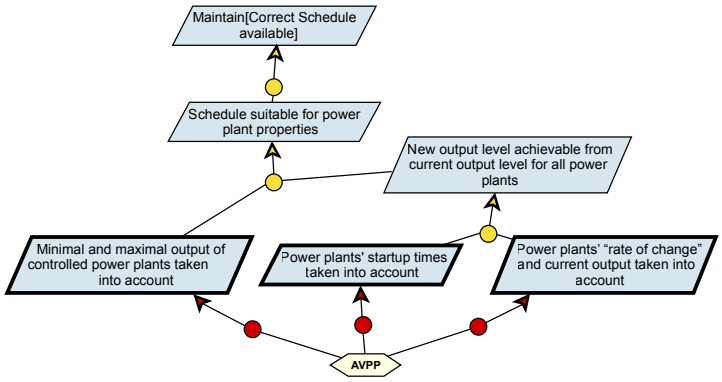


Figure 7.5: Physical power plant constraints as seen from the perspective of the AVPP that creates the schedules. Instead of assigning the requirements constraining the schedule to the physical properties of the power plants directly, the constraints are assigned to the agent that has to adhere to these constraints in the scheduling process.

The introduction of physical constraints hints at an important point: constraints that are observed at runtime can differ from those that define the reconfiguration process. Such processes can be specified as constraint satisfaction problems as outlined in Chapter 9 and the constraints are again be stipulated within the requirements documents. They practically define the controller and the properties valid solutions calculated by the controller have to fulfil. The differentiation is necessary since it might be possible to observe some system parameters at runtime but impossible to use this data for reconfiguration since they are the result of a complex process. For instance, it is possible to observe the actual runtime of the scheduling process, but when looking for a suitable hierarchical structure, it is impractical to simulate different solution candidates and determine the scheduling time. Instead, criteria are used in the formation of the hierarchy that imply that the structure will be able to schedule the power plants within an acceptable amount of time. Such criteria can, again, be expressed as constraints. For the example, they include limits on the number of power plants controlled by one AVPP and for the controllability of each AVPP, i.e., how much controllable power is available and how rapidly this power can be changed from one time step to the next. The specification of the hierarchical structuration process is discussed in more detail in Chapter 9.

[2]Please note that schedules are actually defined over several time steps. Thus the constraint has to hold for all scheduled production targets.

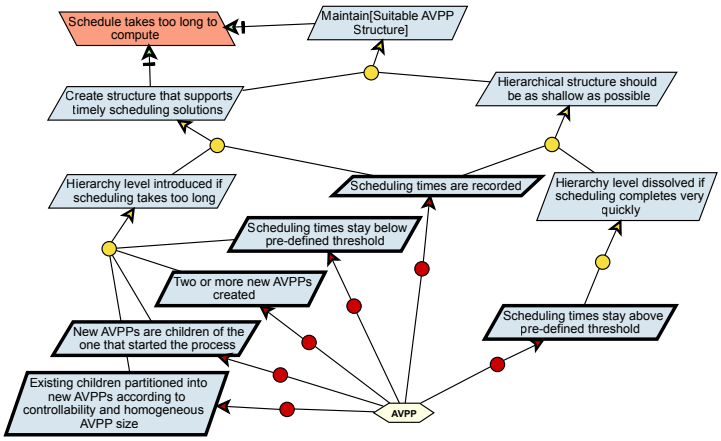


Figure 7.6: Simplified requirements model for hierarchical structuration. Only requirements for the creation of a new hierarchy level are elaborated here. The requirement "Hierarchy level introduced if scheduling takes too long" is translated into a monitoring constraint that triggers the reconfiguration process. The other requirements specify the controller behaviour.

Figure 7.6 shows a simplified version of the requirements model for the top-level goal "Maintain[Suitable AVPP structure]". It is decomposed into two goals that drive the formation process, one that contains the requirements to introduce new hierarchy layers and one that contains the requirements to dissolve hierarchy layers. The former goal is further decomposed into a requirement to record scheduling times and a goal to introduce a layer if the scheduling times exceed a certain threshold. The requirement "Scheduling times stay below pre-defined threshold" captures this in OCL as follows:

```
context AVPP maxspan(3,8) schedulingBelowThreshold:
  self.schedulingTime < self.maximumSchedulingTime
```

Since the constraint is a defined as a MaxSPAN constraint, occasional violations, e.g., due to a fluctuating load on the computer that does the calculation do not cause an immediate reaction. The parameters of the MaxSPAN constraint, the maximum number of violations and the timespan, are not always readily evident but might be the result of simulations or mathematical analysis of the problem.

The other requirements shown in Figure 7.6 do not pertain to the monitoring but are indeed a crude specification of the reconfiguration process. They can be used to express the hierarchical structuration process as a constraint satisfaction problem (CSP). Their use is illustrated in Chapter 9.

7.4 From Constraints to Observer Models

After the relevant requirements have been formally defined with OCL-constraints, the resulting requirements model and the domain model are transformed into an observer model, which is the basis of a specific observer implementation. The transformation is defined in QVT (see sidebar on page 101), which uses *Queries* on the source models to *Transform* them into target models, the *Views*. Part of the views can be pre-specified. The observer model used here has been introduced in Section 6.1. The approach is generic, however, insofar as it is relatively simple to adapt the transformations to a different observer model. Eberhardinger et al. (2013) present, e.g., a transformation to an observer model for an adaptive production cell.

QVT—Query/View/Transformation
QVT (Object Management Group (OMG), 2008) is a model-to-model transformation language, specified by the Object Management Group (OMG) as part of the Meta Object Facilities (MOF). Its main purpose are transformations between models specified in different meta-models with the help of mappings between the classes of the meta-models. *Queries* select parts of the models, *views* provide a way to access more complex excerpts from the models, and *transformations* describe how different constructs are translated.

The newly created transformed model—constructed by the steps outlined below—is described by a UML class diagram, containing the structure of the original system and the Observer/Controller infrastructure and UML activity diagrams. These activity diagrams embed the OCL-constraints in the `holds()` operations of the implementations of the `Constraint` interfaces. This is a necessary intermediate step to retain the information about the constraints. An alternative would be to add them to the `StateModel` class, in whose context they are redefined. However, as they need to be embedded in `holds()`, the additional activity diagrams are used. The activity diagram of `holds()` contains one action node, labelled `return(`ϕ'`)`, where ϕ' is the transformed constraint, interpretable over the state model as an OCL expression without the context, constraint type, and name. Figure 7.7 shows the slightly more complicated definition of the operation for MaxSPAN constraints that stores

Figure 7.7: The activity diagram for the `holds()` operation of a `MaxSpanConstraint`. It first stores the result of the evaluated constraint in the ring buffer and then returns whether the maximum number of violations is exceeded.

the current evaluation of the constraint in the ring buffer and checks the number of entries in the buffer. These static and dynamic diagrams are supplemented by the generic sequence diagram in Figure 6.2 which characterises the interactions of the classes.

Identify Observed Object: First, to generate the observer model it is necessary to identify the classes which should be monitored. These are all classes from the domain model which have an according agent in the requirements model that have a set of semi-formally defined requirements to be monitored. Each class identified in this manner has to implement the `ObservedAgent` interface in the observer model.

Generate Monitoring Structure: For each of the identified agent classes, a specific state model is created by generating a class containing all attributes of the agent class which are required to evaluate the constraints. Additionally an observer derived from the `Observer` class is created for each observed agent. Thus, a relationship between the object that should be observed and the concrete observer is generated. Another relationship is generated between the specific observer object and a generated concrete controller class derived from `Controller`. The controller is left as a step and must be implemented by other means. To complete the control-loop a connection between the generated controller class and the agent class is established.

Generate Constraint Classes: Most importantly, the specific constraint classes are added to the model. For this purpose, the type of constraint is identified, i.e., if it is a hard or a soft constraint by evaluating the keywords **`inv`**, **`soft`**, **`maxspan`**, and **`maxpenalty`** in the OCL statement which indicates the type of the constraint. Once the constraint type is determined, the specific class stub is generated, which implements either `HardConstraint`, `SoftConstraint`, `MaxSpanConstraint`, or `MaxPenaltyConstraint` respectively. Each constraint specified during the requirement analysis that must be monitored at runtime thus has a specialised class. Figure 7.8b therefore contains a `NetworkFrequencyConstraint` class that has been derived from the constraint in Listing 7.2.

If a constraint contains other constraints—such as constraint c4 from Section 6.2—the inner constraints are expanded here by replacing them with the equivalent OCL expression. Thus,

```
context A inv c4: c2 or c3
```

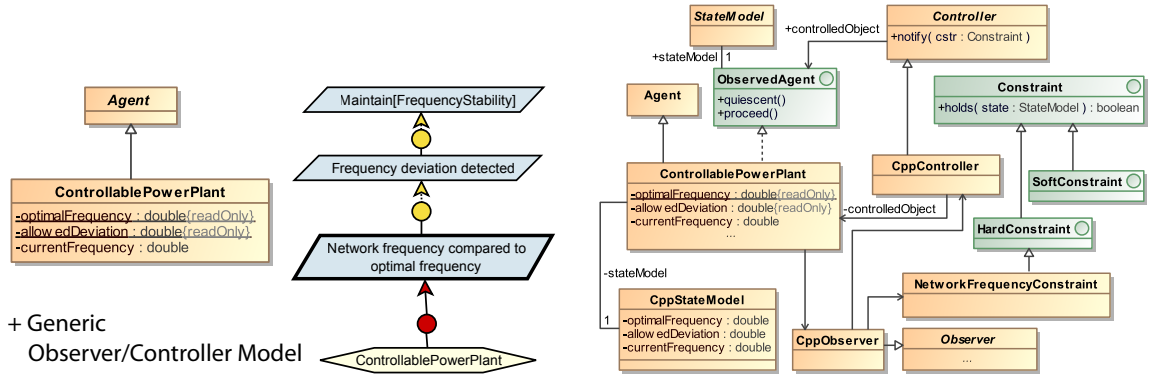
becomes

```
context A inv c4: (y<200) or (z<200)
```

Basic validations are performed at this point, e.g., to check that `c2`, `c3`, and `c4` are all defined within the same context. In addition, the weights for weighted soft constraints as described in Section 5.4 are computed based on the constraint relationships modelled before, yielding `WeightedSoftConstraint` specialisations for the respective constraints.

Generate Dynamic Model: The final step in the creation of the observer model is to extract the statement from the OCL constraint and put it into the guards of the activity diagram that describes the functionality of the `holds()` method of this constraint class. A generic activity diagram for the constraint type is used by copying it and replacing the wildcard with the specific extracted guard. The activity diagrams for MaxSPAN constraints and max penalty constraints can be reused as they are since they are based on evaluating the inner constraints.

This procedure is repeated for every constrained agent. After the completed transformation there is one observer per agent, but there are several constraints per observer. Furthermore, a controller is able to register for a specific constraint by a specific observer. This enables using specialized controllers for optimization and for maintaining correct behaviour according to *INV*. Figure 7.8 shows the elements that are used in this transformation process as well as a simplified version of the resulting class diagram. The transformation creates class diagrams for all agents, as well as the diagrams that model the respective methods for the new constraint classes and the observer. It also checks the domain



(a) Simplified source models for the transformation.

(b) The final, transformed observer model.

Figure 7.8: The sources of the transformation (simplified representation) and the resulting simplified class diagram for the example given in Section 7.2.

model and the requirements model for inconsistencies, e.g., if the requirements model defines agents for which no class exists in the domain model. The result of the transformation process is a platform independent model of the monitoring infrastructure specified in the UML2 meta-model of the Eclipse Modelling Framework (EMF).

The `Controller` class is only a stub at this point, with no functionality. Its specification can be much more complex than the observers' and is usually a part of the design documents. However, the template provided by the transformation detailed here can be used as a starting point for the modelling and the implementation of the controller.

7.5 Transformation from Observer Models to Observer Implementations

The final transformation to actual observer implementations contains many platform specific choices, e.g., whether or not observers and controllers are independent agents or become part of the agents defined in the domain model, whether properties of the agents can be accessed directly or only via message passing, etc. It will therefore have to be adapted to each target platform and target system. An example of a rather non-standard transformation that does not produce source-code but an XML specification of the agents is detailed in (Eberhardinger et al., 2013). Most of the basic principles—outlined in the following—remain the same, regardless of the concrete transformation target. However, the created code is necessarily platform-specific and contains elements for the selected target platform.

Xpand

Xpand[3] is a model-to-text transformation language based on templates and maintained by the Eclipse Model To Text (M2T) project. An Xpand template uses a model as an input and produces text output by replacing placeholders in the template with content from the models. To select model content, an OCL-like expression language is integrated whose syntax resembles Java.

We developed a template which can be adapted for the use in a specific target system. This template is defined in the language Xpand, which is a part of the Eclipse Modelling Framework (EMF) and enables model-to-text transformations. The template contains static parts, written in the target programming language that do not change and only have to be copied into the source code files, including, e.g., initialisation code such as instantiating a ring buffer in case of the Max-SPAN constraint. In addition, the template contains dynamic parts that depend on the elements in the observer model. The code for these dynamic parts is created based on the structure of the class diagrams and the activity diagrams and changes whenever the underlying model changes. The dynamic parts are evaluated when the transformation is performed. This final transformation consists of two steps:

Structural Transformation: First, stubs of the classes contained in the class diagram of the observer model are generated. As defined in the static part of the transformation template, these stubs are integrated into the target platform, i.e., the multi-agent platform or middleware the system will run on and can contain additional initialisation or housekeeping code. The implementation of the

[3] `http://wiki.eclipse.org/Xpand`

generic sequence diagram depicted in Figure 6.2 is also included in the static part and thus has to be included in the template. This step transforms the entire class diagram into system specific source code.

Functional Transformation: Afterwards, the activity diagrams are transformed into implementations of the `holds()` methods in the abstract specialisations of `Constraint`. For this purpose, the OCL statements in the transition-guards have to be translated into conditional statements expressed in the language of the target system. The dynamic part of the template parses the OCL statement into an abstract syntax tree (AST), including the constrained attributes, the OCL functions, logic operators, etc. The AST can be transformed into code for the specific target system. We use the Eclipse OCL grammar as a basis and employ a custom ANTLR parser that supports all standard OCL constructs. In comparison to using a standard parser such as the one provided with Eclipse OCL or Dresden OCL directly, this gives us more flexibility with regard to the target language and thus the transformation target.

Platform-specific Transformations

In concrete terms for the observer model of the autonomous power grid, shown in Figure 7.8, the output of this last step is as follows: The target system is the multi-agent framework Repast Simphony, which is implemented in the programming language Java. So to generate the outline of the class diagram for each interface and class of the observer model, a file containing the basic class declarations, including attributes, method stubs, etc. has to be generated. Next, the `holds()` method of the class `NetworkFrequencyConstraint` is translated into working Java code. For this purpose, the AST created by the OCL parser is translated into Java syntax, e.g., an OCL "`and`" into a Java "`&&`" and OCL methods such as "`includesAll`" into corresponding Java constructs.

Instead of parsing the constraints, it would also be possible to use the faculties of tools like Dresden OCL (see (Demuth and Wilke, 2009) and Section 6.6) to check the constraints at runtime. However, a parser like the one used here can be used to create code for target systems other than those based on a Java Virtual Machine which Dresden OCL is limited to as it uses bytecode weaving. In principle, tools like Dresden OCL do not provide data gathering facilities or infrastructure for interactions with controllers. They can thus be used for constraint checking within the monitoring infrastructure but not as a replacement for the concepts proposed here.

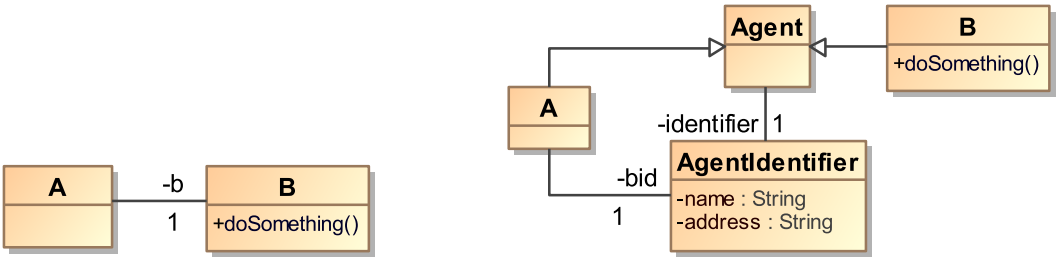
The classes and sequence diagrams have to be translated into the target programming language and the target platform, i.e., the multi-agent platform or middleware the system will run on. Sequence diagrams become implementations of the methods of the classes. The OCL-constraints are parsed to conditional statements within the class-specific implementations of `Constraint.holds()` for the corresponding constraint classes. Since the generic O/C model does not include multiple inheritance or other features that can not be readily transformed into any target language, standard transformations, e.g., for Java can be used.

Multi-agent Specific Issues

Regardless of the concrete programming language, multi-agent system generally impose some peculiarities that have to be regarded in the transformation process. Most importantly, associations between classifiers in a UML class diagram can not be translated as in an object-oriented (OO) approach. In an OO language such as Java or C#, an association such as the one between classes `A` and `B` in Figure 7.9a is usually translated into a field of the class that owns the association end. In the example, `A` would thus have a field titled `b` of type `B`. In practice, `A` thus has a placeholder for a pointer to an instance of `B`. `A` can then access public methods and fields of `A` by simply calling `b.doSomething()`.

In multi-agent systems with message-passing where `A` and `B` are separate agents, however, `A` has no direct access to `B`. In fact, the instances of these two classes can run on computers on different continents. Therefore, if `A` wants to call `b.doSomething()`, it has to send a message to `B`. Thus, a communication endpoint address replaces the pointer to `B`. Figure 7.9b shows how the agent classes `A` and `B` are decoupled in a MAS. `A` no longer has a direct association to `B`, but has an association to an `AgentIdentifier` that holds the address of `B`[4]. The messaging services of the underlying platform can then be used by `A` to send a message, requesting the execution of `doSomething()` to `B`.

[4]In most MAS, a directory service exists to find other agents based on their properties or names. A structuration algorithm can also ensure that agents know each other. The implementation of HiSPADA, e.g., directly sets the agent identifiers for subordinates in AVPPs.



(a) Standard association between classes.

(b) Associations adapted to message-passing MAS.

Figure 7.9: Different modelling options to denote relations between agents in UML class diagram. The use of standard association between agent classes is simple but the implementation will not adhere to the standard semantics ascribed to an association. The more involved version including an agent identifier is truer to the actual implementation but introduces clutter into the models.

Since the transformation from a "standard" UML model to one adapted to a message-passing agent system is straight-forward, the more precise modelling style often omitted. The transformation can either happen at a later point in time or is skipped entirely and becomes part of the transformation to platform code.

Connection to the Controller and Bootstrapping

A concrete observer implementation will have to be coupled with a controller that adapts the system accordingly. The necessary classes have already been introduced with the generic O/C model and an actual controller implementation can be based on this structure. The transformation creates stubs for the appropriate classes and methods such as `activate()`. For the network frequency example discussed earlier, there are a number of decentralised approaches that change the output of power plants to stabilise the network frequency (cf. Section 8.4). Other algorithms—such as HiSPADA (cf. Chapter 4)—can use the same infrastructure as the basis.

An interesting issue is the creation of appropriate bootstrapping code to start observers and controllers along with the respective agents. During the bootstrapping phase, instances have to be created and associations set accordingly. After this initialisation, the controllers will have to register with the observers. Therefore, a phased system boot is usually advised.

7.6 Transformation from Constraints to Rely/Guarantee-Models

As mentioned in Section 7.2, requirements elicitation for a compartmentalised system of systems has some parallels to the specification of rely/guarantees (Jones, 1983; Misra and Chandy, 1981). We have previously used the technique for compositional proofs of the behaviour of adaptive systems (Nafz et al., 2013). By proving properties of the individual agents, global system properties can thus be proven.

The basic idea is that each component guarantees a specific behaviour as long as it can rely on some properties of its environment. The behaviour of a component is specified by a *guarantee* $G(V, V')$ provided by the component. This is expressed as a predicate over the component's transitions. Basically, the guarantee of a component is equivalent to its externally observable behaviour. To be able to guarantee the specified behaviour, the component needs to be able to make assumptions about its environment, as it *relies* on certain—but not necessarily completely specified—aspects of behaviour of its environment. If no relies are formulated at all and the environment is thus completely arbitrary, a component will not be able to give any guarantees, as every system action can immediately be revoked by the environment[5]. To create relies that limit the behaviour of the environment as little as possible, they are usually defined by excluding some particular behaviour. A typical property of the environment is that it does not change a component's internal variables. Formally, a rely $R(V', V'')$ is specified over the environment transitions.

The behavior of a component Ag_i can then be specified using both rely and guarantee. As long as the rely $R_i(V', V'')$ holds, the component guarantees $G_i(V, V')$. This property is formalized as

[5]Note, that from the local view of a component, the environment contains all other components.

$R_i(V', V'') \xrightarrow{+} G_i(V, V')$. The rely/guarantee specification abstracts from the internal implementation of the component and specifies the external behaviour a component should exhibit. It is therefore a black-box specification.

When composing sub-systems in a system of systems, R/Gs define the interface between sub-systems. Thus, constraints and basic R/Gs can be derived from informal requirements at the same time and refined iteratively for use in both runtime monitoring and in formal verification, respectively. The relies and guarantees formulated for agents can be used in verification by applying a compositionality theorem (Bäumler et al., 2011) that describes the correlations between the rely/guarantees of the components and defines the proof obligations for the verification of a global rely/guarantee. A decomposition into the adaptive system part and the functional system part has been suggested in (Nafz et al., 2013) that allows the verification of the Observer/Controller specification and of the functional system separately.

As the invariant in the RIA is the combination of the constraints, it is not surprising that the generic guarantees for a centralised Observer/Controller include this invariant as shown below. The guarantee also includes the assurance that the O/C does not interfere in the functional system and leaves the variables of the functional system unchanged. For a distributed O/C architecture (cf. Section 3.3), each O/C guarantee would contain the part of the invariant it has to ensure but essentially look the same otherwise. An O/C relies on the agents not to stop a reconfiguration process by themselves.

$$
\begin{aligned}
G_{o/c}(V_{all}, V'_{all}) :\leftrightarrow \quad & (\forall i \ : \ (Ag_i.reconf \wedge \neg Ag'_i.reconf \rightarrow INV_{RIA}(V'_{all}))) \\
\wedge \ & (\forall i \ : \ \neg Ag'_i.reconf \rightarrow noInterference(V_{all}, V'_{all})) \\
\wedge \ & (Prop(V_{all}) \rightarrow Prop(V'_{all})) \\
\wedge \ & Unchg_{sys}(V_{func} \setminus V_{conf})
\end{aligned}
$$

The relies and guarantees of the agents describe that the agent relies on the O/C not to alter any variables as long as no reconfiguration is taking place. It guarantees that it will adhere to the quiescent state if prescribed by the controller and not leave the reconfiguration by itself.

The variable parts of these definitions are the invariant INV_{RIA} and the property $Prop(V_{all})$. The global invariant and the invariants for the individual O/Cs in case of a distributed architecture can be directly derived from the requirements documents. The property the system has to fulfil corresponds to the system goal, e.g., the stability of the power grid frequency. While not explicitly formulated for the transformation process shown so far, system goals can also be described formally in the same manner as requirements. They are not, however, limited to variables observable for individual agents but can take a broader view of the system.

For these reasons it seems natural to provide a transformation from the requirements documents to a basic rely/guarantee specification that can be used in formal verification of the system. This requires, however, a translation from the OCL constraints used in the specification of constraints so far to a formal logic (the one used in the example is ITL$^+$ Bäumler et al., 2010) but other logics are possible as long as a compositionality theorem exists. Alternatively, the constraints could be specified in OCL and in ITL$^+$ in parallel. Such an approach would, however, increase the effort during analysis and design and increase the risk of inconsistencies between the two specifications. We thus propose the creation of appropriate transformation rules that allow the automatic creation of an R/G specification from the constraints formulated in OCL or a subset thereof. A mapping of OCL constructs to temporal logic has to be found for this purpose. The work of the SecureMDD project (Moebius et al., 2012) can serve as a starting point for this endeavour.

Chapter Summary and Outlook

This chapter detailed a transformation process that creates an implementation of observers and controllers for a specific system based on the requirements and the generic Observer/Controller model introduced in Chapter 6. The requirements modelling step is extended with specification of constraints that form the basis of both the behavioural corridors monitored at runtime as well as the adaptation processes (cf. Chapter 9). The transformation of different types of hard and soft constraints is shown and possibilities to adapt the transformation for specific environments are discussed. While the discussion here focusses on the observation of constraints, the created structure can also serve as the foundation for controller implementations. Those implementations can be based on constraint satisfaction and optimisation techniques as discussed in Part IV.

Part IV

Organised Stabilisation

Algorithmic techniques to ensure that system goals can be achieved, esp. in hierarchical systems where the combination and abstraction of control models are necessary.

Chapter 8

Stabilisation and Adaptation in Adaptive Systems with Feedback Loops

Summary. Feedback loops are a basic principle for purposeful, active behaviour (Rosenblueth et al., 1943). They are widely used in control theory and are at the heart of the Observer/Controller architecture. This chapter introduces the principles of feedback loops and shows how they allow systems to evolve towards stable equilibria states. A discussion of how these faculties can be harnessed in open, heterogeneous self-organising systems and how system structure and goals influence these processes is complemented with an example for a negative feedback loop from the power management domain and its function as a stabilising force within the system. Finally, an outlook on how interfering feedback loops can jeopardise stability provides insight into problems that can occur when several feedback loops are at work at the same time.

Large-scale open self-organising systems have a complex internal dynamic: they are characterised by processes that adapt to the environment, change the system structure, and make autonomous decisions to reach the goals of the individual agents and, ideally, of the system as a whole. These processes are running concurrently in different sections of the system, triggered by agents that react to violations of constraints as a reaction to environmental changes, new information, or interactions and state changes. In a way, the constant exchange of information between the system components and its environment are similar to *far-from-equilibrium systems*, systems that exchange energy or matter with the environment continually. These systems can exhibit self-organisation (Nicolis, 1989) that leads to the spontaneous formation of structure or patterns. As system designers, we are not only interested in the formation of structure. We want the system to adapt by exploiting all degrees of freedom at its disposal to achieve purposeful behaviour (Rosenblueth et al., 1943). These degrees of freedom are the "free variables" of the system that can be changed by adaptation processes according to the constraints specifying the corridor of correct behaviour.

Adaptation as a reaction to changes in the system or its environment is enacted by *feedback loops*. A feedback loop uses information coming from the system to influence its future course (its "feeding back" information about itself to the system). Together with information from the environment and from other system components, decisions can be made to achieve a desirable state. Feedback loops have been recognised as the principle instrument to describe, analyse, and design the dynamics of adaptive and self-organising systems (Brun et al., 2009). The observer/controller architecture and the MAPE-cycle embody such feedback loops, as discussed in Chapter 3. These architectures can effect different kinds of feedback that can serve different purposes. *Positive feedback* can drive adaptation processes by exacerbating changes in the system. *Negative feedback* can dampen changes to stabilise the system in its current state. Both kinds of feedback and their application in large-scale open self-organising systems are discussed in Section 8.1. Congenial to the constraint specification approach defined in page 57, adaptation and stabilisation problems can be specified as constraint satisfaction optimisation problems, as described in Section 8.2. Of course, the structure of the system plays an important part in the ability of a system to adapt and to reach its goals as detailed in Section 8.3. While this thesis is mainly concerned with adaptation with positive feedback that transforms the system into a new stable state, negative feedback that stabilises the system is outlined in Section 8.4. Finally, Section 8.5 discusses the interference of different feedback loops and the consequences of such disturbances.

8.1 Negative and Positive Feedback for Stabilisation and Adaptation

In its most simple form, a *feedback loop* exists whenever a system changes the parameters of the system (action) based on observations about itself and the environment (percept). A system consisting of a heater and a thermostat, e.g., changes the temperature of the room according to observations of that same parameter. If the room gets cooler, the heater heats up, if the room gets warmer, the heater shuts down. A more complex setting is a social situation where an individual adapts its behaviour according to the "feedback" it gets from its conversation partner. If the partner is responsive to a certain line of argument, e.g., the individual uses that information to convey her ideas in a similar way in future discussions. Even more complex settings involve a number of intertwined feedback loops influencing one or more variables that are mutually dependent on each other. An example for such a complex settings are financial markets in which a number of factors, including the current interest rate, economic growth, employment rate, risk aversion, political developments, and others, influence each other. A simple example from this domain are bank runs, i.e., sudden mass withdrawals of saving deposits that can cause the collapse of a bank since its financial assets are depleted. Bank runs can occur because rumors about the liquidity of a bank can cause depositors to panic and withdraw their funds for fear of losing them in case the bank goes bankrupt. This behaviour can trigger other depositors to withdraw their savings from fear that the bank run itself will cause the collapse of the bank. The end result is a kind of swarm behaviour in which each individual tries to protect its own assets without regard of the overall system stability.

Control Theory and Feedback Loops
Feedback loops are widely used in control theory as closed-loop control systems that feed the output of the system back into the controller. They are usually employed to achieve a stable system state. This stable state is often an equilibrium in the sense that significant effort is required to change to another stable system state. The main task of the controller is to maintain that state and avoid oscillations.

A bank run is an example of a *positive feedback loop*, a process that causes massive changes in a system due to its self-reinforcing nature. A small trigger can lead to a system change that, in turn, causes more changes. Other examples of such feedback loops are the run-away chain reaction in a nuclear fission bomb or the stimulated emission of light in a laser. As can be seen in the example of the bank run, positive feedback can have undesirable side-effects. It can cause emergent behaviour that—while rational from the individual point of view—causes cataclysmic effects on the system level. Therefore, the emergent behaviour of the system has to be observed (cf. Chapter 6) and a balancing element has to be introduced in the system.

This balancing element often comes in the form of a *negative feedback loop*, a process that dampens changes in a system due to its self-correcting nature. It can thus prevent run-away reactions as well as oscillations and over-compensation. In the case of the bank run, negative feedback could be provided by governments ensuring that deposits will be paid back at all times. This breaks the positive feedback causing the bank run and thus stabilising the system. Such stimuli can also come from inside the system as well. A sustained nuclear chain reaction in a fission reactor, e.g., is only prevented from causing a fatal nuclear meltdown by careful regulation and absorption of neutrons with graphite moderators.

Feedback Loops in Adaptive and Self-organising Systems

The architectural principles of the MAPE-cycle and the Observer/Controller introduced in Chapter 3 embody feedback loops and allow their use in adaptive and self-organising systems. They allow the system to observe itself and its environment and react to changes accordingly. These structures are independent of the kind of feedback they implement and both kinds of feedback are indeed used. As Figure 8.1 illustrates, positive feedback is used to align the the components in the system to achieve a stable state or equilibrium. Negative feedback maintains this stable state. The system changes its equilibrium through a process of positive feedback in case drastic external forces are applied (Heylighen et al., 2001). Positive feedback is associated with self-organisation, i.e., changes in the structure of the system. Negative feedback, on the other hand, is associated with adaptation, i.e., changes in the parameters of the system. Although the distinction is often not as clear cut in real systems, this classification can be helpful when identifying and describing the different processes that act in the system concurrently.

Borrowing from thermodynamics, the different states depicted in Figure 8.1 can also be described as "phases" in which the system can be. Switching between a stable state and different kinds of feedback can thus be denoted as a *phase transition*. These terms are often used in self-organising computing

systems since a lot of the early research on self-organisation stems from physics and chemistry (see, e.g., Prigogine and Nicolis, 1977). According to Heylighen et al. (2001), self-organisation drives a system towards an equilibrium state. Phase transitions occur when the "amount of energy" the system receives from the outside changes. When applying such terms to computing, however, they have to be reinterpreted. "Amount of energy" can, e.g., be the information provided to the system from the outside or changes in the output of a stochastic process. In terms of the autonomous power management system, changes from the outside can be new power plants, changes in the credibility of individual power plants due to, e.g., intermittent hardware failures, or changes in the load the system has to provide.

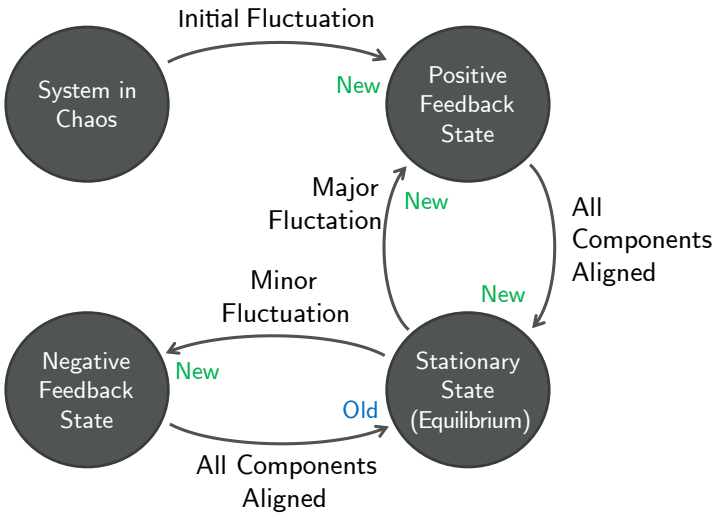


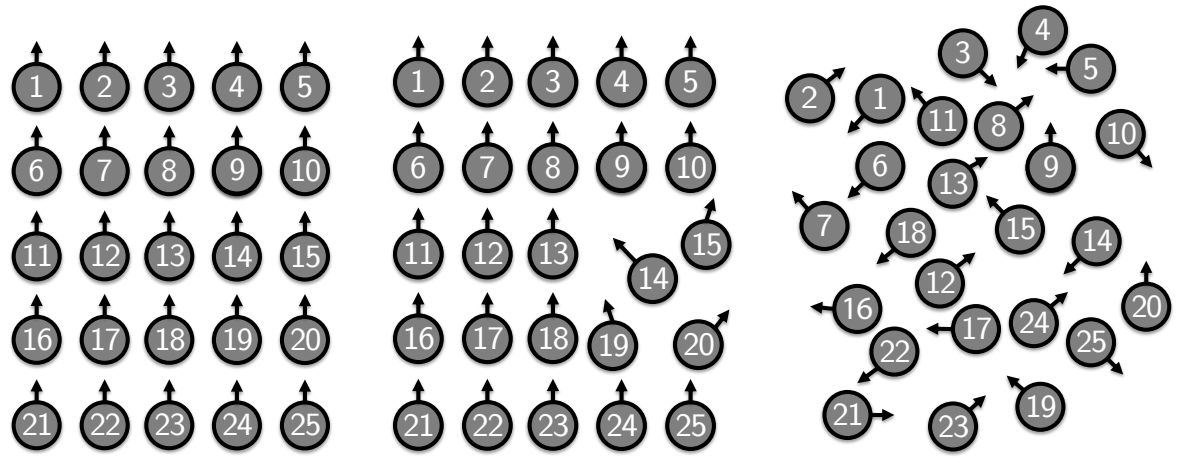
Figure 8.1: Stabilisation in adaptive systems with positive and negative feedback loops. A system in an initial chaotic state is driven towards a stable state by a process of positive feedback. Once this equilibrium is reached, negative feedback stabilises it and dampens smaller fluctuations. In case drastic fluctuations occur in the system, positive feedback realigns the components and transfers the system into a new stable state. Based on Heylighen et al. (2001).

The different kinds of feedback can be achieved by different means in a adaptive and self-organising system. In addition, as Figure 8.1 indicates, different feedback loops are necessary to balance each other out. Positive feedback alters the system structure and thus involves a self-organisation process. Such processes often run on a long time scale as finding a good system structure can be time consuming. They are also often located on the meso or macro levels of analysis since they influence more abstract parts of the system and require some regional knowledge. The hierarchy formation algorithm HiSPADA—detailed in Chapter 4—can thus be seen as providing positive feedback within the system. To ensure its termination, a "moderator" is needed. Such moderators are also useful to avoid that stable states are left in case of small fluctuations.

Homeostasis
In biological systems, homeostasis is the property of maintaining a variable, e.g., temperature or the pH-level, within very narrow bounds (Reimann, 1996). The processes that maintain these narrow bounds are feedback loops that self-regulate these variables.

Negative feedback can provide such moderators and can be achieved with reactive algorithms that locally change parameters of the agents (cf. Section 8.4), thus trying to balance changes in the environment or in the system itself. The autonomous frequency stabilisation algorithm used in the autonomous power management case study is an example for such a mechanism. It runs on a relatively small time scale, making changes to the system without the need of much computation time or coordination between entities. Negative feedback often works on the micro level, enacting changes on the individual agents. Sometimes, however, the negative feedback is built into the observation and control environment for a positive feedback loop. In case of HiSPADA, e.g., the process terminates automatically after a new hierarchy level has been introduced, an existing one has been dissolved, or a hierarchy level has been reorganised. Due to a cooldown time, the structure then stays stable for a while to evaluate its fitness for purpose. Only if the new structure proves to be unfit, the process is started again. A careful choice of constraints and the use of soft-constraints such as MaxSPAN(cf. Chapter 6) can also provide dampening to positive feedback processes.

A simple system of mobile robots is used to illustrate the different kinds of feedback in a self-organising system. These robots coordinate to form a simple grid pattern as depicted in Figure 8.2a in which the robots are aligned and face the same direction, as indicated by the arrows. When an external disturbance occurs, e.g., because one of the robots is pushed into its neighbours, most of the structure remains intact as shown in Figure 8.2b. The affected robots can realign themselves with their neighbours by a negative feedback process that dampens the change in their position relative to their neighbours. In case of a drastic external force applied to the robots—such as using a broom or



(a) Undisturbed system.

(b) System disturbed by a slight fluctuation.

(c) System disturbed by drastic external forces.

Figure 8.2: A system of simple robots, programmed to align themselves in a grid pattern. The robots primary direction of movement (the orientation of its axis) is indicated by an arrow. If a slight fluctuation occurs in the system, negative feedback can dampen the movement of the robots that were disturbed and realign the system. In case of a drastic fluctuation, the robots will try to coordinate to reshape the pattern and in this process exacerbate the movements induced from the outside. Robots that have not been disturbed might be included in such a positive feedback.

similar implement to jostle the entire ensemble around—the original stable state can no longer easily be restored as depicted in Figure 8.2c. A process of positive feedback kicks in that amplifies the relative changes in position and enables the robots to find a new structure. If robots were left undisturbed, they may be included in the process and the entire ensemble can start changing positions relative to each other. The final outcome of this process is a new, stable state in which the robots are again aligned in a grid, albeit in another order and at another absolute position.

Feedback Loops and Behavioural Corridors

Behavioural corridors, as introduced in Chapter 6, provide the means for triggering phase transitions in adaptive and self-organising systems. Whenever a constraint is violated, the controllers that are informed about the violation can trigger a adaptation or a self-organisation algorithm in an effort to stabilise the system. A general rule of thumb is that negative feedback tries to keep the system within the corridor while positive feedback occurs when the system leaves the corridor. This implies that negative feedback is thus triggered by the violation of soft constraints, while positive feedback is triggered by the violation of hard constraints. On the other hand, the corridors also provides a means to detect when a phase transition to the stable state has occurred: as soon as the constraints are satisfied again, the feedback processes can terminate.

8.2 Specification of Adaptation Processes as Constraint Optimisation Problems

A further relation to the behavioural corridors specified with the Restore Invariant Approach is the reuse of constraints to specify the problem that an adaptation algorithm has to solve. Since the constraints define correct system behaviour, a positive or negative feedback process that changes the structure or the configuration of the system must compute solutions that adhere to these constraints. Therefore, correct configurations of the system are solutions to *constraint satisfaction problems* (CSPs)—defined by the relevant constraints that are also part of the invariant—and the algorithms employed thus solve such a problem. The use of constraint solvers and other algorithms for this purpose is discussed in the sidebar on page 113.

In case soft constraints are specified, the corridor of correct behaviour allows to further introduce a measure of optimality. Soft constraints can be used in the reconfiguration of a system incorporating them

into the solution process. The constraint satisfaction problem then becomes a *constraint satisfaction optimisation problem* (CSOP) in which a solution with a high quality is sought. The quality is defined by an optimisation or *objective function*, basically a metric that allows comparing different solutions with regard to their suitability. Solving a CSOP with the objective of minimising a delta to an optimum can be used as a heuristic to solve problems in the system class regarded in this thesis. Intuitively, the "optimum" is the middle of the corridor, or any point within the optimal area within the corridor as defined by the soft constraints. In practice, however, the objective function often adds an additional metric that goes beyond what has been specified by constraints. In any case, adaptation processes can be described as finding solutions to constraint optimisation problems defined by the same constraints used in the observation of emergent behaviour and specified in the requirements engineering as outlined in Chapter 7.

An important point is that there is usually no single optimal solution in any situation the system can find itself in. Instead, the algorithm solving the CSOP has to find a *pareto-optimal solution*. This is always the case if more than one free variable is influenced by the solution and different criteria on these variables are defined. Constraint Relationships are a tool that can be helpful in looking for this solution and can guide the solution process. The technical details of using soft constraints to specify CSOPs that are solved in large-scale open self-organising systems are discussed in Chapter 9.

The constraints that are part of the invariant are not, however, always sufficient as a specification of the CSOP the system has to solve. Consider the autonomous power management example: the physical limitations of the individual power plants, e.g., their minimal and maximal production, are not observed at runtime since it is a property of the power plants that these will not be violated. When calculating a schedule for the power plants, however, these restrictions have to be taken into account. It is therefore necessary to distinguish constraints into those that are part of the invariant (keywords **`inv`**, **`maxspan`**, and **`maxpenalty`**), soft constraint (keyword **`soft`**), and physical restrictions (keyword **`phy`**). For a more detailed discussion of this aspect, please refer to Section 7.3.

Constraint Satisfaction in Self-Organising Systems

It has to be noted that, while it is useful to describe the reconfiguration of a self-organising system as a constraint satisfaction model and it is possible to use constraint solvers for this purpose (cf. Nafz et al., 2009b), actual solution processes can use other approaches, e.g., heuristic algorithms or distributed coordination-based algorithms. HiSPADA, introduced in Chapter 4, is an example for such a distributed algorithm, solving a CSP that defines correct structures without using a classical constraint solver. A more thorough discussion of CSP solvers in self-organising systems can be found in Section 9.4.

8.3 System structure and system goal

The purpose of the system is to achieve its goals. If the system achieves this by means of self-reflective behaviour, it is deemed intrinsically purposeful (Rosenblueth et al., 1943). However, it is usually not the goal of a system to achieve a certain structure. Instead, it is the goal of the system to achieve a structure that suits its purpose. Thus, the structure the system self-organises should be suitable for achieving the system goals, i.e., the structure should be suitable so that those algorithms that operate within the structure are able to achieve their goals. For instance, hierarchical scheduling for power plants depends on the system structure, as an AVPP creates schedules for the underlying power plants. If the structure is suitable, high-quality schedules that minimise the difference between the demand and the production are found within a reasonable amount of time. If the structure is unsuitable, it is either impossible to find a good solutions since, e.g., there are not enough controllable power plants available to distribute the load on, or scheduling takes too long since there are too many controllable power plants. The structuration process thus has to include metrics that allow it to create a structure that enables high-quality, high-performance scheduling. In this sense, self-organisation processes provide a supporting role for the system.

A structured system also allows compartmentalisation of adaptation processes. Adaptation processes that stabilise the system can occur in all sub-systems concurrently. Structure allows to limit the effects of the processes to a certain region within the system, thus preventing changes from spreading. Such an approach is useful, e.g., if different sub-systems work under different operating conditions. Negative feedback processes can then adapt parameters for a sub-set of the agents, suited for the situation these agents are in, but with regional knowledge of the operating environment. Thus, the system goals can be achieved in a slightly different way or with a different set of parameters in the individual sub-systems, tailored to the specific needs and environment. The hierarchical structure introduced by the AVPPs in

the autonomous power management example illustrates this compartmentalisation: each AVPP adapts the power plant schedules according to the regional knowledge it collected.

While all stabilisation mechanisms should be independent of the concrete structure that has been established in the system, it is possible that a mechanism relies on the existence of **some** structure to make it, e.g., computationally feasible to find a correct solution within a reasonable amount of time as in the scheduling example. This is an important requirement for the adaptation processes: they should assume as little as possible about the concrete structure they find in a system. This is—at least partially—due to the different time scales under which adaptation process operate. While the system structure should change infrequently, other adaptations occur more often, are even regular occurrences (again, power plant scheduling provides a good example), or work permanently in the background (such as the frequency stabilisation mechanism introduced in Chapter 6).

On the other hand, changes in the system structure, such as a hierarchy, can effect the equilibrium already attained. If a system is stable with regard to one degree of freedom, the adaptation of another degree of freedom can influence the system's ability to operate in a stable fashion. This is, e.g., the case if an adaptation—especially a structural one—can be triggered by the violation of constraints that observe different properties. HiSPADA is triggered either by a violation of timing constraints or by the violation of application-specific constraints, as outlined in Chapter 4. A similar credibility of neighbouring AVPPs constitutes such an application-specific constraint. If the current structure is stable with regard to scheduling times, it might still be possible that the credibility mix changes and a reconfiguration is triggered that causes the system structure to change. This can jeopardise the stable scheduling times and cause subsequent adaptations that change the hierarchical structure of the system.

This latter example illustrates how the inability to reach an equilibrium can in turn effect the system structure. Similarly, if the current structure does not enable the system to find solutions fast enough, the structure will be adapted. It is important to note that the interaction between the constraints outlined here can constitute an interference that might lead to oscillations in the system if negative feedback is not present that dampens these changes. The system could, e.g., detect a situation in which conflicting constraints lead to a cyclic reorganisation and change threshold values, effectively "widening the corridor". If that is not possible, the task of tuning the parameters will have to be delegated to the system designers.

8.4 Stabilisation with Negative Feedback: Local, reactive mechanisms

As mentioned above, negative feedback processes can always work locally, by reacting to a stimulus, e.g., changes in parameters, that can be observed by individual agents. An example for such a process is frequency stabilisation in autonomous power management systems whose requirements and the accompanying monitoring infrastructure were discussed in Chapter 6. While the observation of the network frequency and the necessary observer has been explained there, this section deals with the controller part, i.e., the actual reaction to an observed violation of the constraint that detects deviations from the optimal network frequency. The details of the algorithm as well as a different approach using communication between the agents can be found in (Anders et al., 2012a).

Spinning reserve in current power management systems
Current power management systems use a *spinning reserve* as the primary control to adapt to changes in the network frequency. In the European power grid UCTE, about 3GW of power are reserved for this purpose. They are provided by large power plants that deliberately operate below their optimal output and can be ramped up quickly to compensate deviations within 30 seconds. Each utility is obligated to participate in this scheme.

The goal of the frequency stabilisation mechanism is to maintain a relatively stable power grid frequency of 50 Hz. Since the European power grid is synchronous, all power generators in Europe are required to feed power into the grid with a current alternating at this frequency. For thermal power plants such as nuclear, coal, and biogas, this means that the turbines have to rotate at speeds that produce current at 50Hz. Solar power plants use power inverters to generate an alternating current with the respective frequency. Some power generators also use synchronisation equipment to achieve the frequency.

If demand is higher than production, the frequency drops since the resistance on the side of the power grid rises so that a constant input of energy—in case of a thermal power plant the energy used to heat the steam—will cause the turbines to rotate more slowly than required. Therefore, the energy input into the system must be increased. Reversely, if demand is lower than production, the resistance on the side of the power grid sinks and the turbines rotate faster, causing an increase in frequency at constant energy levels. The influence of the

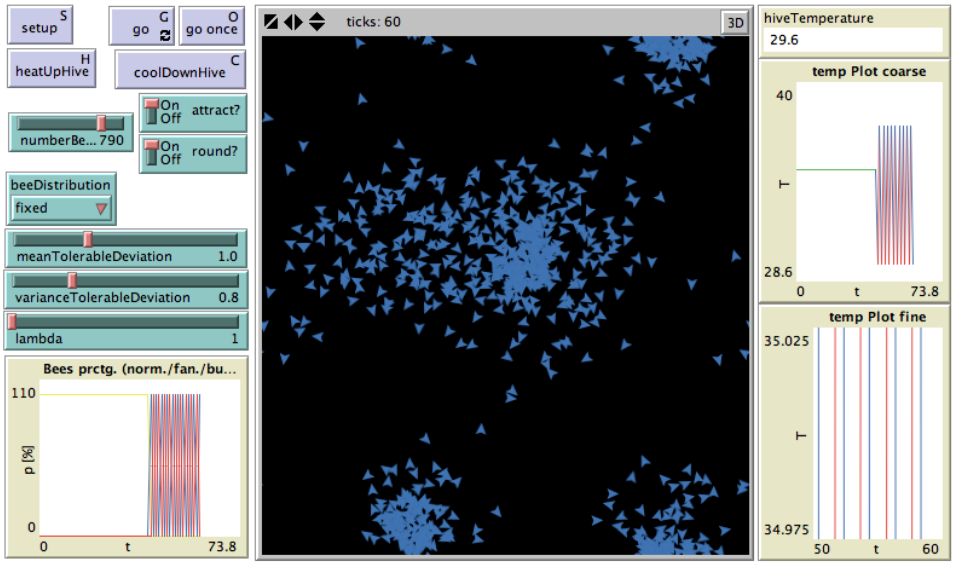


Figure 8.3: The user interface of the bee hive simulation. The bees are depicted as triangles that move and act within the hive. A temperature oscillation can be seen in the graph on the right hand side.

difference of demand and production on the network frequency is determined by the inertia of the system, so that small differences only have minuscule effects on the frequency. Still, frequency stabilisation measures are even performed when the frequency changes only within a few tenths of a percent.

The frequency stabilisation mechanism uses this connection to autonomously adapt the power output of controllable power plants by a small degree to achieve a stable network frequency. Notably, in comparison to current approaches (see sidebar on page 114), it includes *all* controllable power generators into the scheme, distributing the necessary reserves over a much larger number of generators and allowing the system to unburden high-voltage transmission lines. It is a strictly local algorithm that works only with what each individual agent can perceive and is purely reactive since it only takes action if a frequency constraint is violated.

Over-compensation and Oscillations

A naive approach to the problem is to have all controllable power generators react to changes in the network frequency at the same time. A system-wide threshold could be defined that all generators use to detect deviations. If the frequency of the network falls below the optimal frequency minus the threshold, they ramp up their power output. If it rises above the optimal frequency plus the threshold, they reduce their output.

To illustrate the issues with this approach, we use a simulation of an equivalent problem. In bee hives, the optimal breeding temperature lies—depending on the concrete species of bee—around 35 degrees celsius. If the hive cools down too much, bees heat it by "buzzing" with their bodies, thus producing frictional heat. If the hive becomes too warm, bees use their wings to fan cool air into the hive. A simple simulation of this system has been created in Netlogo (Wilensky, 1999)[1]. It allows to specify the number of bees in the hive as well as the thresholds at which they react to temperature changes. Figure 8.3 shows a screen shot of the simulation during a run.

As in the frequency stabilisation problem, all participants (bees and power generators respectively) are sensitive to changes in a shared variable (the temperature and network frequency respectively). The agents also react to a simple constraint limiting the deviation of the shared variable from an optimum. Each agent then reacts the same way, the bees by either starting to buzz or to fan, the power generators by raising or lowering their output.

The result of the naive approach is depicted in Figure 8.4. Since all bees react to a temperature deviation at the same threshold and at the same time, the system enters a state of oscillation. The joint

[1]The concrete model used here has been developed on the basis of a model created by Thomas Kohler as part of the lecture on Self-Organising Adaptive Systems in the winter term 2011/2012. Re-used with permission of the original author.

action of all agents results in an over-compensation that causes the system to react with an adverse action. Intuitively, as soon as the hive gets too hot, all bees start to fan, cooling the hive too much. This causes the violation of the lower boundary constraint, causing the bees to heat. The heating action then puts the hive temperature above the upper boundary, causing the bees to cool and so on.

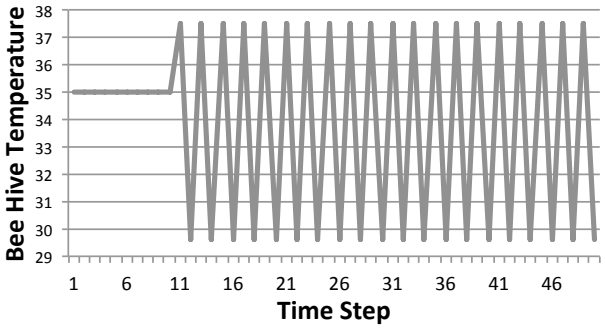


Figure 8.4: Oscillations in the bee hive temperature regulation model. Model settings are the same as in Figure 8.3, i.e., 790 bees with a temperature threshold of 1 degree react to deviations from the optimal temperature by changing the temperature by 0.01 degrees per time step. The system oscillates between 29.6 and 37.5 degrees. The process was started by an external event in time step 10 that heated the system up by 2.5 degrees.

Each bee only changes the hive temperature by a very small amount (by 0.01 degrees). However, as the action of the bees is immediate and synchronised, the joint change in the temperature is significant—in the case of the experiment conducted here, the temperature changes by 7.9 degrees due to the bees reaction. This explains why this unwanted behaviour does not occur with smaller populations: the joint change is much less. In particular, if the joint change returns the system to within the boundaries of acceptable temperatures, no oscillations occur. As it is not possible to reduce the overall number of participants in such a system, oscillations have to be avoided by pursuing other routes.

One possibility is to coordinate the action of the agents by communicating with each other. This way, agents could come to an agreement which agents react to the stimulus. This is a viable way of mitigating over-compensation and Anders et al. (2012a) have shown that the frequency stabilisation problem can be tackled this way. In principle, the agents find a solution to a consensus problem by solving a distributed constraint satisfaction problem. However, communication is time-consuming and requires a suitable infrastructure. In particular, the agents have to know each other and exchange messages directly or access a common black board on which messages are exchanged. Run times can be very high due to the number of messages that must be exchanged and guaranteeing eventual consensus is difficult. Thus, communication is not well-suited to situations in which speed is of the essence.

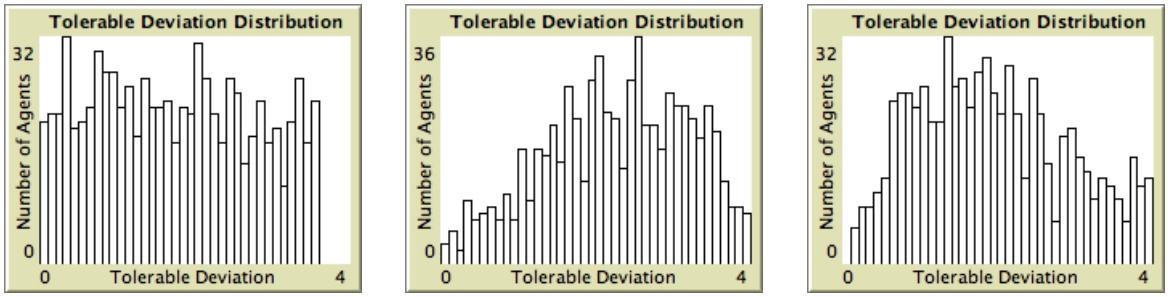
Heterogeneity Promotes Stability

A more promising approach is to exploit *heterogeneity*. Jones et al. (2004) have shown that heterogeneity leads to stable configurations and avoids over-compensation and thus oscillations in bee hives. The principle has been operationalised for self-organising systems by Campbell et al. (2011) who show that heterogeneity—or inter-agent variation as the authors call it—can indeed improve stability. (Anders et al., 2012a) adapted the bee-hive behaviour described by Jones et al. (2004) for use as a network frequency stabilisation mechanism.

The principle is simple: instead of letting all agents react to the external stimulus at the same time, the reaction is staged by using individual thresholds for the agents. This allows integrating all agents into the scheme but limits the number of agents that react to a deviation from the network frequency synchronously. At the same time the number of agents reacting scales with the severity of the deviation. As the thresholds are assigned based on a continuous probability distribution, some agents react faster to deviations than others. Agents on the edges of the distribution have a high tolerance but will participate in case of large deviations. For frequency stabilisation, it is advisable to assign high thresholds to powerful generators since small deviations can easily be tackled with a swarm of small generators, each contributing only a small correcting amount of power.

The success of this approach thus hinges critically on the type of probability distribution used and its parametrisation, especially with regard to the expected deviation that has to be compensated and the mean value of the distribution. Figure 8.5 shows examples of probability distributions used in the experiments conducted with the bee hive. Since the deviation that was expected is a sudden change in temperature of 2.5 degrees, the mean value μ was set close to this value at 2.4. This yields acceptable results with a quick stabilisation.

The evaluation results should only be regarded as a proof of concept. Anders et al. (2012a) offer much more thorough evaluations of the behaviour of the frequency stabilisation algorithm. However,



(a) Uniform distribution. (b) Gaussian distribution. (c) Gamma distribution.

Figure 8.5: Examples for distributions used in the experiments. The uniform distribution is created by a uniform random process limited to double values between 0 and $\mu + \sigma^2$. The gaussian distribution is created with $\mu = 2.4$ and $\sigma^2 = 1.2$. The gamma distribution is created with $\mu = 2.4$ and $\lambda = 1$. Gaussian and gamma distributions show better results as fewer agents react to small deviations. Values are indeed continuous but have been rounded for display in the histogram.

the results shown in Figure 8.6 show some interesting properties. First of all, the uniform distribution does not prevent oscillations with the chosen parameter settings. This is due to the fact that a relatively large number of agents reacts to deviations, even when they are low. Again, the same argument as above applies: as long as the number of agents reacting synchronously to the deviation and their aggregated response to it exceeds the threshold after which an adverse response is triggered, the system will oscillate. To avoid this, the parameters could have been selected so that the distribution is spread further and the number of agents reacting to small deviations is lower.

The gaussian and gamma distributions avoid this. They show stabilising behaviour in almost all simulation runs. The gaussian distribution is the most stable with the chosen parameter set, due to the fact that a relatively small number of agents are involved for small deviations and many agents are clustered around the mean value in the typical bell curve. On the one hand that means the graduated response to small deviations as outlined earlier. On the other hand, if the deviation becomes large, the agents respond together. The gamma distribution shows a slower stabilisation behaviour and does not always stabilise the system. This is due to the chosen λ parameter that skews the distribution so that many agents react to relatively small deviations. If λ is chosen more sensibly, the gamma distribution yields good results as well.

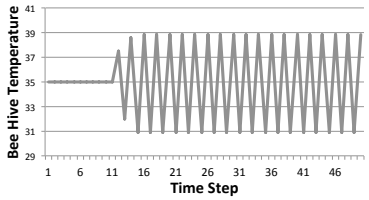
For the gamma distribution, a transient oscillating behaviour can be observed. This is due to the fact that the distribution causes a strong response first, that over-compensates. The over-compensation, however, is less severe than the original stimulus. Thus, less agents react to it. This way, the stabilisation process takes a couple of steps, gradually subsiding over time. The same behaviour is the reason why the gamma distribution sometimes exacerbates the problem: if the distribution is sufficiently skewed, the initial reaction will be more drastic than the original stimulus, forcing more agents to participate in stabilisation, and so on. This constitutes a *positive feedback loop*—and of course should be avoided for a mechanism designed to provide negative feedback.

8.5 Interference of Feedback Loops

Unfortunately, different feedback loops do not necessarily work together very well. Especially concurrent positive feedback processes can have side effects on each other if they operate on the same degree of freedom. As an example, consider the HiSPADA algorithm detailed in Section 4.4 and the concurrent adaptations it can perform. If two AVPPs detect a violation of the application-specific constraints that describe a suitable hierarchy, they will start both start an adaptation of the hierarchy. HiSPADA takes great care to avoid these concurrent adaptations from clashing, especially through the use of the `isDissolvable` and `canReorganise` predicates. As soon as the reconfiguration starts, all agents in the neighbourhood of the AVPP are no longer available for concurrent reconfigurations. This effectively prevents interference.

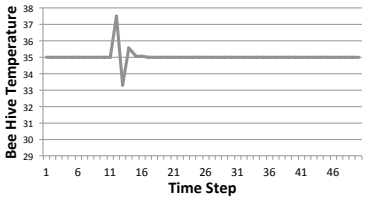
However, these measures are a specific solution to the problem of interference in one specific algorithm. In case independent feedback processes work in the system concurrently, such explicitly designed countermeasures might not be applicable. In general, whenever feedback loops work on the

No. of runs with oscillations: 100
Avg. time till stabilisation: N/A



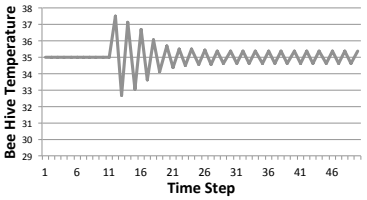
(a) Uniform distribution.

No. of runs with oscillations: 0
Avg. time till stabilisation: 5.43



(b) Gaussian distribution.

No. of runs with oscillations: 15
Avg. time till stabilisation: 11.28



(c) Gamma distribution.

Figure 8.6: Evaluation results for the bee hive temperature regulation mechanism for different probability distributions. All results are averaged over 100 runs. The parameters were chosen to be $\mu = 2.4$, $\sigma^2 = 1.2$, and $\lambda = 1$. Only the gaussian distribution reliably avoids oscillations and stabilises the system fastest. The average time till stabilisation is an over-estimation since the time till a final, stable temperature is reached is measured. In many cases, however, the changes in temperature before reaching this equilibrium are minuscule.

same degrees of freedom, interference is possible. To detect and analyse such dependencies between adaptation mechanisms, identifying the degrees of freedom that are influenced by an algorithm and the degrees of freedom the algorithm depends on is crucial: if a feedback loop changes a degree of freedom another loop depends on, interference can occur.

Such a dependency is present in the power management system: the scheduling process depends on the system structure since the compartmentalisation of the system influences the quality of the schedules and the degrees of freedom available to the scheduler. If the system structure is adapted, the schedules become invalid since the power plants they have been created for might no longer part of the compartment. On the other hand, if the quality of the schedules for the current system structure is unsatisfactory, the structure must be changed. Since the AVPPs are nested, even reorganisations on a lower hierarchy level that are completely opaque to higher levels can influence the quality of schedules.

Additional work will be required to fully understand the interferences that occur, how to detect them at runtime, and how to deal with them, especially as the systems using feedback loops and adaptation mechanisms become more complex and more diverse feedback loops are implemented. First steps in this direction have already been made (see, e.g., Eze et al., 2013; Sanderson et al., 2013), but principled investigations of the general mechanims of interference are still pending.

Chapter Summary and Outlook

This chapter introduced the notion of feedback loops as a foundational mechanism for adaptation and stabilisation in adaptive and self-organising systems. They can be used to describe and analyse phenomena in natural, physical, economical, and other systems and are used in control theory to design control systems. The different feedback states play a crucial role in triggering adaptation and stabilisation mechanisms and can be used to describe adaptation process in the context of technical systems as well. An example from the domain of power management shows how feedback loops occuring in natural systems can serve as the basis for technical solutions and how simple principles can be transferred between the domains. The Observer/Controller architecture embody feedback loops in self-organising systems and constraint satisfaction and optimisation approaches as described in the following chapters provide opportunities to implement both negative and positive feedback.

Chapter 9

Stabilisation with Positive Feedback: Constraint Satisfaction with Soft Constraints

Summary. Using the formalism for soft constraint problems introduced in Chapter 5, this chapter outlines how constraint satisfaction techniques can serve as the main positive feedback loop for stabilisation in large-scale open self-organising systems. Based on the scheduling problem from the power management domain, we show how constraint satisfaction problems can be formulated whose solutions conform with the behavioural corridor specified during requirements analysis as shown in Chapter 6 and observed with the architectures described in Chapter 7. We also show how these models can be extended to incorporate optimisation criteria and measures of uncertainty.

Publication. Some of the concepts used in this chapter have been described in (Steghöfer and Reif, 2012) and (Anders et al., 2013a).

As outlined before, positive feedback is necessary when the system becomes unstable and a drastic change is required to maintain system stability. This is correlated with the violation of hard constraints as the violation indicates a development that impedes the correct functionality of the system and thus a violation of the corridor of correct behaviour. The monitoring infrastructure introduced in page 57 allows to detect violations of constraints and introduces the necessary support for the integration of controllers and staged reactions.

Modelling a reconfiguration task as a constraint satisfaction problem (CSP) is an intuitive and efficient way to describe valid system configurations and thus specify the controller. In the final system, specialised algorithms can assume the task of reconfiguring the system. However, formalising the CSP as a solvable model and testing the capabilities of the model can be an important step in the design of the algorithm. First of all, it requires the designer to think about the constraints under which the controller has to operate and that describe valid solutions. This provides feedback on requirements elicitation and on the system design, as early experiments with the CSP can reveal that a problem is over-constrained and thus no solutions can be found or that it is under-constrained and the solutions are different than expected. In these cases, the requirements have to be checked and the domain or design models might need to be augmented. Second of all, it allows to establish a "gold standard" for the reconfiguration with which the performance of other algorithms can be measured. As a standard constraint solver is not distributed, works with global knowledge and with algorithms that are guaranteed to be optimal—at the price of a potentially unlimited runtime—they find solutions that can serve as benchmarks. Solutions found by a distributed algorithm with local knowledge, possibly a heuristic, could then be compared with the optimal solutions. The acceptability of the alternative algorithm can thus be defined by the difference of the solutions it creates to the optimum.

In case the Restore Invariant Approach or a similar technique is used and requirements are expressed as constraints as described in Chapter 7, parts of the CSP required for reconfiguration are already available. We distinguish monitoring constraints and reconfiguration constraints (Nafz et al., 2009b): monitoring constraints are relevant for the observation of correct behaviour (cf. Chapter 6)

while reconfiguration constraints are relevant for the specification of the reconfiguration algorithm. Some constraints serve both purposes.

This work is an extension of the research done by Nafz et al. since we are now able to specify problems with soft constraints and apply them to hierarchical systems. Additionally, the connection between the specification and the requirements has been developed and the integration into a software engineering approach is now possible. Existing work details the specification of CSPs for reconfiguration in non-hierarchical Organic Computing systems (Nafz et al., 2009b, 2011) and the use of constraints as a result checker (Fischer et al., 2011) to validate the results of heuristic algorithms. In this work, constraint models are localised to accommodate the compartmentalisation of systems of systems and thus a hierarchical system structure with black boxes. Indeed, many reconfiguration problems that are solved by algorithms from the literature can be expressed as CSPs and the algorithms proposed in the literature can thus be regarded as problem-specific (distributed) CSP solvers.

Section 9.1 introduces examples for CSPs in the autonomous power management case study and exemplifies the feedback between requirements engineering and controller design with the example of constructing the model for a partitioning of power plants. The use of constraint relationships to express preferences over constraints and thus optimise the solutions is illustrated in Section 9.2. As uncertainty plays an important role in open self-organising systems, Section 9.3 outlines the use of trust and especially trust-based scenarios in CSPs. Finally, specialised solution algorithms for CSPs that do not make use of specialised constraint satisfaction techniques are discussed in Section 9.4.

9.1 Constraint Satisfaction Problems in Autonomous Power Management

In the following, we will introduce different CSP formulations for three different problems that appear in autonomous power management systems. The first problem is the partitioning of power plants within a hierarchy level or in a flat system. Basically, it is the specification of the SPADA algorithm, introduced in Section 4.1. It serves as an explanation of the iterative process of creating a solvable CSP from the requirements and refining the requirements to capture all necessary constraints. The second example is the creation of schedules for individual power plants controlled by an AVPP. It outlines how CSP models can evolve and how adding features iteratively makes the models more expressive and more complex. Finally, the formation of hierarchies with HiSPADA is formulated as a CSP to demonstrate the difference between constraints for monitoring and constraints for reconfiguration.

All three cases shown in the following implement the control part of a positive feedback loop: the initial impulse that triggers these control and adaptation decisions may be small, such as the violation of one of the partition composition constraints or the detection that the current schedule is no longer suitable for the current situation. However, the impulse is reinforced by the changes that occur due to these control decisions. This is obvious if the decentralised versions of SPADA and HiSPADA are used but is also true for the central solution of the scheduling problems: existing schedules are modified by the system in an attempt to find new ones that allow to deal with the new situations, thereby altering the existing schedules for a potentially large number of power plants.

Partitioning of Power Plants

The partitioning of power plants into AVPPs reduces the complexity of the scheduling process and groups dissimilar power plants together while the AVPPs themselves should be similar to each other. This anti-clustering balances controllability and uncertainty within the AVPPs while providing an externally stable and homogeneous behaviour to the outside. SPADA (cf. Section 4.1) creates such an anti-clustering in a self-organised manner based on the data provided by the power plants. The partitioning process is started whenever the observer of an AVPP detects the violation of one of the constraints that specify correct partitions (the “application-specific constraints” in the case SPADA is used to drive a hierarchical partitioning with HiSPADA as explained in Chapter 4. This initial impulse is reinforced by the system and propagates in a positive feedback process that leads the system into a new stable state.

An initial requirements model that captures the basic requirements for such a process is depicted in Figure 9.2. It introduces the goals to, at the one hand, create AVPPs that are similar when compared to each other and, on the other hand, are heterogeneous internally. The requirements derived from these goals deal with the provision and collection of the information required to perform comparisons of similarity, with the exchange of power plants between AVPPs in case AVPPs are not similar, and

with the comparison of the mixture. This latter constraint "Compare mixture of energy sources and uncertainty with other AVPPs" will be observed at runtime and can thus be expressed as a monitoring constraint in OCL. The domain model, depicted in Figure 9.1, must make it possible to access the relevant data and to navigate to the other AVPPs.

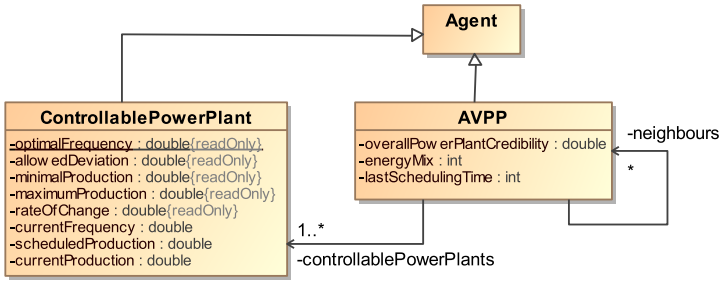


Figure 9.1: The simplified domain model for the autonomous power management case study. AVPPs are connected through a `neighbours` association and contain properties for the relevant fields used in the OCL constraints.

```
context AVPP inv energyMix:
  self.neighbours->forall (n : AVPP | (self.energyMix - n.energyMix).abs <= 3)

context AVPP inv credibilityMix:
  self.neighbours->forall (n : AVPP |
    (self.overallPowerPlantCredibility - n.overallPowerPlantCredibility).abs <= 0.2)
```

Listing 9.1: Monitoring constraints to ensure the similarity of AVPPs, defined in OCL.

The OCL constraints define similarity by limiting the difference between the measures. The energy mix is simplified to an integer value by assigning a stochastic energy source such a solar power plant 1 and a controllable power plant -1. This allows applying simple arithmetic operations to describe the mix inside an AVPP. An AVPP with a value of less than 0 consists predominantly of controllable power plants, one with a value greater than 0 of stochastic power plants. In practice, most AVPPs will have values far greater than 0 due to the wide deployment solar power plants. The simplification, however, allows to compare AVPPs of arbitrary mix with each other.

Initial Problem Specification The constraints in Listing 9.1 are monitoring and reconfiguration constraints since any partitioning of power plants into AVPPs will have to adhere to these constraints. Therefore, the same formulation can be reused in a specification of the reconfiguration problem. They thus serve as a starting point for a CSP model, expressed in the following in OPL—the Optimisation Programming Language—used with the IBM ILOG CPLEX Optimisation Suite. However, a number of additional specifications are required to define a solvable CSP model.

First, the data structure that holds the data of the individual power plants has to be defined. OPL offers a tuple data type that can hold any number of simple data types (integers, strings, floats, sets, arrays). The PowerPlantData tuple contains a credibility value and an energy source, encoded the same way as in the OCL constraint.

```
tuple PowerPlantData {
  key string name;
  int energySource; // -1 is controllable, 1 is stochastic
  float credibility;
};
```

The data for the power plants is defined in an external file. This allows us to switch configurations quickly and testing the CSP model with different initial configurations of power plants.

```
{PowerPlantData} PowerPlants = ...;
```

Now, a first technical speciality of the CSP model has to be considered. For the CSP solver, the simplest way to solve the problem is to create exactly one AVPP. As there are no other AVPPs, the mixture is perfect. We prevent this by introducing a minimal number of AVPPs that have to be created. A second particularity is the fact that we have to limit the maximal number of AVPPs the solver can create. This is due to the fact that we have to work on finite domains and that the number of possible solutions increases exponentially with the number of AVPPs the solver could theoretically create. Thus,

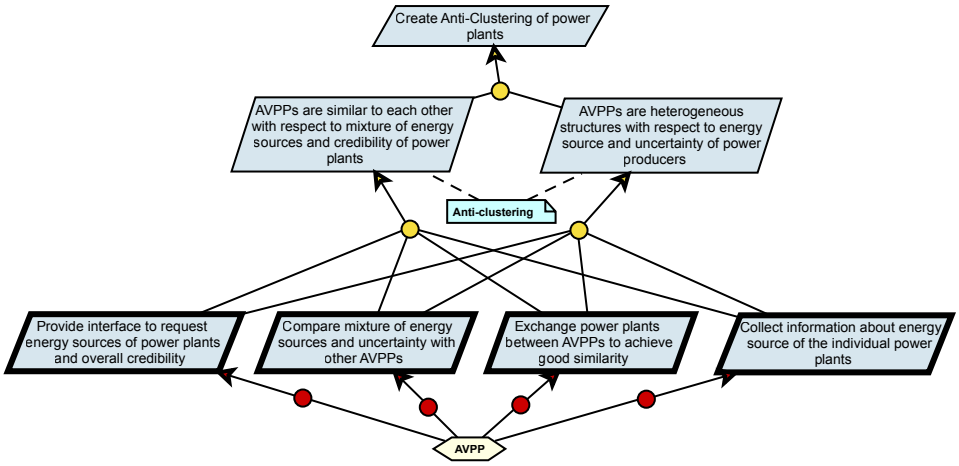


Figure 9.2: Initial requirements for an anti-clustering of power plants into AVPPs. The goals prescribe external similarity and internal heterogeneity with respect to the energy sources of the power plants and the credibility of the power plants.

to keep running times within a limit, we give an upper bound as well. In addition, we define a range that can be used in expressing array bounds later on.

```
int minAvpps = 2;
int maxAvpps = 6;
range AVPPS = 1..maxAvpps;
```

Next, the decision variable is defined. The decision variable is the free variable that the solver tries to assign values to. In our case, the membership[PowerPlants] array holds an integer value for each of the power plants. The value determines in which AVPP the power plant is.

```
dvar int membership[PowerPlants];
```

A number of decision expressions follow. These special kinds of statements allow using decision variables in more complex expressions. This is helpful since it can greatly simplify the way constraints are expressed. The first expression defines a sparse matrix that holds a boolean value (the integers 0 and 1 correspond to the boolean values true and false. It can be used to check if a power plant belongs to a certain AVPP, given by the number of the AVPP within the admissible range.

```
// Sparse matrix for the membership of power plants to AVPPs
dexpr int avppMembership[i in AVPPS][p in PowerPlants] =
  membership[p] == i;
```

The next expression calculates the size of each of the AVPPs by summing the entries of a single row in the membership matrix. The row is given by the number of the AVPP and has to be within the admissible range.

```
// Array of the number of members of AVPPs
dexpr int avppSize[i in AVPPS] =
  sum ( p in PowerPlants ) avppMembership[i][p];
```

The avppPopulated array contains a boolean value indicating whether the AVPP given by its number has any members.

```
// Array indicating whether an AVPP has members
dexpr int avppPopulated[i in AVPPS] =
  avppSize[i] >= 1;
```

The expression avppCredibility gives the overall credibility of the AVPP calculated by dividing the sum of the credibility of the power plants by the number of power plants in the AVPP. It uses the membership matrix to multiply it with the credibility of each power plant. Since the membership matrix only contains a 1 if the power plant is part of the AVPP, only power plants within the AVPP are counted.

```
// Array of the avg. credibility of AVPPs
dexpr float avppCredibility[i in AVPPS] =
  (sum (p in PowerPlants) p.credibility * avppMembership[i][p]) / avppSize[i];
```

The energy mix of an AVPP is calculated similarly by a simple summation. The same rationale as above applies.

```
// Array indicating the energy mix of AVPPs
dexpr int avppEnergyMix[i in AVPPS] =
  sum (p in PowerPlants) p.energySource * avppMembership[i][p];
```

Now that all definitions have been prepared, the actual constraints are defined. Thankfully, due to the careful choice of decision expressions, the constraints themselves are rather simple. Before the two constraints defined in the requirements model are formulated, however, it is necessary to define two boundaries for the solver. First of all, a power plant can only be part of one AVPP. Second of all, the number of AVPPs created has to be within the boundaries defined before.

```
subject to {

  // Technical stuff
  forall (p in PowerPlants) {
    limitMembership: 1 <= membership[p] <= maxAvpps;
    enforceAvppCount: minAvpps <= sum(i in AVPPS) avppPopulated[i] <= maxAvpps;
  }
```

The constraints of Listing 9.1 now translate into rather simple expressions.

```
  // Constraints for the comparison between AVPPs
  forall (a,b in AVPPS) {
    energySourceSimilarity: abs(avppEnergyMix[a] - avppEnergyMix[b]) <= 2;
     credibilitySimilarity : abs(avppCredibility[a] - avppCredibility[b]) <= 0.2;
  }
```

This model is solvable with CPLEX. It contains the constraints defined in the requirements models so far and is able to find a partitioning of power plants into AVPPs. The way these AVPPs are expressed—by a membership matrix and with simple integers—may seem counter-intuitive at first, but is a rather natural way of modelling in a constraint model.

The model is tested with a very basic configuration with a total of 30 power plants, 15 controllable power plants and 15 stochastic ones. The solution expected is a partitioning in which the power plants are evenly distributed to a number of AVPPs between the minimum and maximum defined. The solution CPLEX produces, however, is quite different. The solver creates exactly 2 AVPPs, and populates the first one with 29 out of the 30 power plants while putting one power plant in the second AVPP. It is obvious that this solution is undesirable, but it fulfils all constraints. The AVPPs are similar within the boundaries given by energySourceSimilarity and credibilitySimilarity and the minimal number of AVPPs has been created as well. Clearly, the current requirements are insufficient.

First Revision: Incorporate AVPP Size An additional factor that has to be considered in partitioning for the controllability of an AVPP is the number of controllable power plants in an AVPP. It is desirable to have a sufficient number of controllable power plants and thus of controllable power to offset fluctuations in the production of stochastic power plants. A first idea is thus to make the size of the AVPPs similar. Together with the energy mix constraint, this ensures that the AVPP has a number of controllable plants and a number of stochastic ones. An additional requirement in the KAOS model and an additional OPL constraint allows the exploration of this approach.

```
  // Constraint for the comparison of AVPP sizes
  forall (a,b in AVPPS) {
     sizeSimilarity : abs(avppSize[a] - avppSize[b]) <= 2;
  }
```

Second Revision: Incorporate Aggregated Rate of Change Indeed, the solution produced by the solver is much closer to what was intended. The solver now creates 6 AVPPs, 4 of which contain 6 power plants and 2 of which contain 4 power plants. Energy mix and credibility mix constraints are fulfilled. However, on closer inspection, the result is not applicable to the actual system. This is due to the fact that the simplified model does not yet consider the output of the power plants. The output of the controllable plants within an AVPP, however, should be balanced with the output of the stochastic power plants. Only then is it possible to deal with fluctuations effectively within the AVPP. Therefore, the requirements have to be extended to reflect a balance in the power output instead of in the size. Indeed, the relevant factor is the rate of change of a power plant, i.e., how much the output of a controllable power plant can change within a certain amount of time.

The updated requirements model is shown in Figure 9.3. It now contains additional goals and requirements for the controllability of each AVPP. In essence, AVPPs should have a similar amount of power that they can adjust per minute. The changes can be reflected in the constraint satisfaction model by introducing a rate of change for the power plants and by adding the respective constraints. The PowerPlantData tuple is extended by an additional field rateOfChange:

```
tuple PowerPlantData {
  key string name;
  int energySource; // -1 is controllabe , 1 is stochastic
  float credibility ;
  float rateOfChange; // The adjustable output per minute
};
```

An additional decision expression that sums up the rate of change in each AVPP is introduced.

```
// Array of the aggregate rate of change of an AVPP
dexpr float avppRateOfChange[i in AVPPS] =
  sum (p in PowerPlants) p.rateOfChange * avppMembership[i][p];
```

Finally, the constraint ensuring the similarity of the AVPPs is incorporated.

```
  // Constraint for the comparison of AVPP's rate of change
  forall (a,b in AVPPS) {
    rateOfChangeSimilarity: abs(avppRateOfChange[a] - avppRateOfChange[b]) <= 30;
  }
```

The result the solver produces are now very close to what a modeller would expect. The refined model can thus serve as a benchmark for other algorithmic solutions. Further refinements could be made, e.g., to make the rate of change of each AVPP dependent on the aggregated power output of the stochastic power plants. Such extensions of the model could, again, feed back into the requirements engineering process where they would be captured in the model. The example illustrates how the iterative-incremental process can be driven by simple, prototypical implementations of the adaptation algorithms as constraint satisfaction models.

A drawback of the final model is that the constraints contain numeric boundaries for the allowable differences between AVPPs. A final escalation could thus be to change the model into a constraint satisfaction optimisation problem in which the differences between AVPPs are to be minimised. Indeed, the SPADA implementation does not use bounds by tries to minimise the difference between AVPPs by evaluating how switching power plants between AVPPs influences the mix. The decisions are, however, based on the three factors identified during requirements engineering, namely energy source, credibility, and rate of change.

A Simple Power Plant Scheduling Model in OPL

We use a simplified formalisation of the power plant scheduling problem from (Nafz et al., 2013). The total load—i.e., the power demand—which should be met is denoted as L_c and is known to the AVPP. The AVPP calculates schedules for the power plants based on a load prognosis L_{prog} which approximates the future load. Each power plant $i \in N$ has a scheduled target output $P_{target,i}$ that is derived from the AVPP's target output which in turn is determined by the predicted load ($P_{target} = L_{prog} = \sum_i^N P_{target,i}$). As the schedule is made for several timesteps in advance, P_{target}, L_{prog} and $P_{target,i}$ are lists of values. The scheduled target output for time t is denoted by $P^t_{target,i}$.

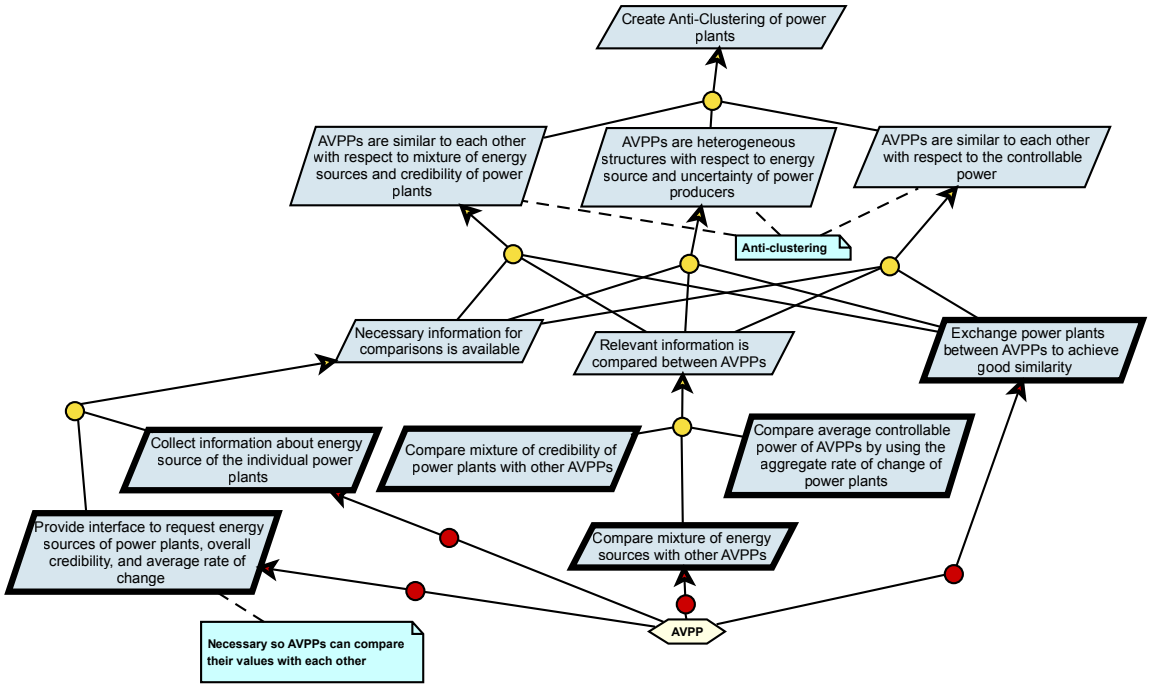


Figure 9.3: Requirements model with additional requirements for the controllability of an AVPP. In comparison to Figure 9.2, an additional goal to cluster the AVPPs according to a similarity of their controllable power has been introduced. An according requirement for this comparison has been introduced as well. In addition, two new goals that deal with the necessary information for the comparisons and with the comparisons themselves have been introduced to give the requirements more structure. Such a refinement is a typical result of an iterative-incremental requirements elicitation approach in which the designer learn about the system while working with the customer and with early prototypes.

The property that is of importance in the energy system is grid stability. The power grid is sensitive to imbalances between consumption and production. If they differ, the network frequency changes which can lead to power outages and destroy equipment. Therefore, the AVPP has to ensure that—if the forecast of the upcoming load is good enough—it will always produce as much power as requested. This boils down to an approximate equality between the scheduled target output of the power plants for the current timestep and their actual output in this time step:

$$gridStability(V_{all}) := P^{now}_{target} \approx \sum_{i}^{N} P_{actual,i}$$

It is not sensible to demand strict equality since the prognosis can never be guaranteed to be exactly equal to the actual load. As there is a band in which the power grid can operate and the reactive mechanism can compensate slight deviations, this is not strictly necessary.

A valid solution to the scheduling problem is one that describes a valid schedule for the system, ensures that in sum as much power is produced as currently is consumed, and that the schedule will be able to cover the consumption predicted for each timestep t.

The first constraint describes that a plant's assigned target output has to be either zero or between the power plant's minimal and maximal output possibilities.

$$C_{cons} : \forall i, t : P^t_{target,i} \neq 0 \rightarrow P_{min,i} \leq P^t_{target,i} \leq P_{max,i}$$

Further a valid schedule has to assure that the change of output power from one time step to the next is not greater than the rate of change of a plant (v_i).

$$C_{change} : \forall i, t : |P^{t+1}_{target,i} - P^t_{target,i}| \leq v_i$$

In every timestep each plant's target output should be approximately equal to the current output ($t = now$).[1]

$$C_{balanced} := \forall i : P^{now}_{target,i} \approx P_{actual,i}$$

The mathematical formulation of the power plant scheduling problem can now be translated into an equivalent model in OPL, solvable by the IBM ILOG CPLEX Optimisation Suite. First, the input data and its structure is defined. A time range, indicating the scheduling horizon is defined first. The different kinds of power plants are identified by name, and for each type, a list of power plant names will be loaded from a separate data file. This is indicated by "...".

```
range TIMERANGE = 0..95;

{string} ControllablePlants = ...;
{string} SolarPlants = ...;
{string} WindPlants = ...;
```

Next, a data structure containing data of the different power plants, including their maximal and minimal production, their rate of change, and the production cost is defined.

```
tuple powerPlantData {
  float minimalProduction;
  float maximalProduction;
  float rateOfChange;
  float pricePerKWh;
};
```

These data structures are used to populate an associative array mapping the power plants to their respective data. Again, "..." indicates that the data will be loaded from a separate file.

```
powerPlantData PowerPlant[ControllablePlants] = ...;
powerPlantData SolarPlant[SolarPlants] = ...;
powerPlantData WindPlant[WindPlants] = ...;
```

The demand (energyConsumption), as well as the solar radiation and the wind strength are loaded from the data file as well. These are indexed over the previously defined time range, meaning that for each point in time for which a schedule is created, one data point will be available.

```
float energyConsumption[TIMERANGE] = ...;
float solarRadiationFactor[TIMERANGE] = ...;
float windStrengthFactor[TIMERANGE] = ...;
```

Now, the *decision variables* are defined. They are equivalent to the free variables in the mathematical formulation and define the dimensions of a solution.

```
// The scheduled energy production per controllable plant in each time step.
dvar float+ energyProduction[ControllablePlants][TIMERANGE];

// The amount of energy missing or superfluous to meet the demand in each time step.
dvar float violation[TIMERANGE];
```

A decision expression—a complex term that contains decision variables—is used to make formulation of the constraints easier. It calculates the total energy production at each point in time, combining production from solar, wind, and controllable power plants.

```
dexpr float totalProduction[t in TIMERANGE] =
  (sum( p in ControllablePlants ) (energyProduction[p][t]) +
  sum ( p in SolarPlants ) (solarRadiationFactor[t] * SolarPlant[p].maximalProduction ) +
  sum ( p in WindPlants ) (windStrengthFactor[t] * WindPlant[p].maximalProduction ));
```

The objective function is to minimise the violation, i.e., the difference between production and demand in each time step. The overall violation is added up over all time steps, but taking its absolute value ensures that a negative violation does not balance out a positive one.

[1] For the verification this is formalized as the difference may not exceed a certain ϵ.

```
minimize
  sum ( t in TIMERANGE ) abs(violation[t]);
```

Now, the constraints start, introduced by the **subject to** keyword. All controllable power plants must operate within the operational limits of their maximum and minimum production (C_{cons}).

```
subject to {
  forall ( p in ControllablePlants ) {
    forall ( t in TIMERANGE ) {
      // Each plants production must always be greater than its minimal production
      minimalProductionConstraint:
        PowerPlant[p].minimalProduction <= energyProduction[p][t];
      // Each plants production may never exceed its maximal production
      maximalProductionConstraint:
        energyProduction[p][t] <= PowerPlant[p].maximalProduction;
    }
```

The constraint for the rate of change (C_{change}) is a little more complicated since it compares the production in the current timestep t with the production in the next time step t+1. To avoid overflow errors, the time range regarded must thus be limited. This is achieved by using t **in** 0..**card**(TIMERANGE)−2:

```
    forall ( t in 0..card(TIMERANGE)-2 ) {
      // The percentual change in output must always be lower than the rate of change
      rateOfChangeConstraint:
        abs(energyProduction[p][t] - energyProduction[p][t+1]) <=
          PowerPlant[p].maximalProduction * PowerPlant[p].rateOfChange;
    }
  }
```

Finally, the requirement that demand and production have to be in relation to each other ($C_{balance}$) has to be formulated. This is where the violation—which must be minimised according to the objective function—comes into play as well.

```
  forall ( t in TIMERANGE ) {
    // Demand must always be satisfied ( violation[t] allows deviation from full satisfaction )
    satisfyDemandConstraint:
      totalProduction[t] + violation[t] == energyConsumption[t];
  }
}
```

The satisfyDemandConstraint allows a violation to occur. Since it is minimised, the solver will ensure that the violations are as small as possible while not violating any other constraints. The satisfaction of the demand can thus be seen as a soft constraint, while all other constraints are hard and have to hold at all times.

Most of the constraints that are part of this model have been captured during the requirements elicitation process as physical constraints of the power plants.

First Revision: Incorporating Power Plant Shutdown So far, the model only considers permanently running power plants. A power plant thus always produces a certain minimum of power. In a real systems, some power plants can be switched off and put into a standby state in which they can be reactivated quickly. To reflect this in the model, a new decision variable is introduced that indicates whether a power plant is running or not.

```
// Determines whether or not a power plant is running in each time step.
dvar boolean powerPlantRunning[ControllablePlants][CONSIDERED_RANGE] in 0..1;
```

The new decision variable makes it necessary to adapt some of the constraints. A power plant can either be switched off or produce power. Therefore, the value for powerPlantRunning has to be considered in the maximum and minimum output constraints.

```
// Each plants production must always be greater than its minimal production
minimalProductionConstraint:
  (powerPlantRunning[p][t] == true) =>
```

```
    PowerPlant[p].minimalProduction <= energyProduction[p][t];
// Each plants production may never exceed its maximal production
maximalProductionConstraint:
  energyProduction[p][t] <= PowerPlant[p].maximalProduction;
```

Furthermore, a constraint has to be introduced to enforce that a power plant has no output while it is not running.

```
// If a power plant is not running, its output has to be zero
notRunningNoOutputConstraint:
  (powerPlantRunning[p][t] == false) => (energyProduction[p][t] == 0);
```

The rate of change constraint has to be adapted similarly.

```
// The percentual change in output must always be lower than the rate of change
rateOfChangeConstraint:
  powerPlantRunning[p][t] == 1 =>
    abs(energyProduction[p][t] - energyProduction[p][t+1]) <=
      energyProduction[p][t] * PowerPlant[p].rateOfChange;
```

These constraints are sufficient to give the solver an additional degree of freedom and creating solutions in which individual plants can be deactivated. This simplified model assumes that it is possible to switch off any controllable power plant. It also assumes that startup is immediate. To incorporate startup times and minimal standby times, additional measures have to be taken.

Second Revision: Standby and Startup Times The standby of power plants can be differentiated in a hot and a cold standby. The terms indicate the origin of the differentiation from thermal power plants where the temperature of the thermal system defines the time until the system becomes operational again. Hot standby allows re-starting the plant quickly. In cold standby, the thermal systems have to be re-heated before power can again be produced. In many cases, this process requires an external power source, thus making a starting power plant that comes back from cold standby a consumer. The models presented here do not take this into account, however.

Incorporating standby and startup makes the problem much more complex since the effects of starting a power plant are not evident in the next time step. In addition, switching a power plant on or off incurs a form of commitment since the decision can not easily be undone. These aspects have to be considered in the scheduling process.

First of all, the power plant data has to be extended with the appropriate fields to hold information about the standby and startup behaviour. The new fields includes indicators how long the power plant has to be running or has to be stopped consecutively. These restrictions avoid thrashing, i.e., the constant starting and stopping of power plants and depend on the type of plant. A running water power plant can, e.g., be started and stopped relatively quickly by changing the inflow of water while an inert system like a thermal power plant is not as easily powered up and down. The other elements of the tuple will be described later.

```
tuple powerPlantData {
  float minimalProduction;
  float maximalProduction;
  float rateOfChange;
  float pricePerKWh;
  int minOffTime;           // minimal timesteps the plant has to be off
  int minOnTime;            // minimal timesteps the plant has to be on
  int numBPs;               // number of breakpoints for piecewise linear function
  float fAtZero;            // function value at 0
  int countDownInit;        // initial count down if plant was about to start
  int consRunningTimeInit; // the consecutive running time at initialisation
  int consStoppingTimeInit; // the consecutive time the plant has been stopped at initialisation
};
```

The main tool to incorporate hot and cold standby is a *piecewise linear function* (PWL) that is defined for each power plant and that returns the startup time for a previous stop time. The function is defined over breakpoints and slopes. A PWL is composed of line sections with constant slopes. At the breakpoints, the behaviour of the function changes. In the example, the PWL captures the change

in startup times after the power plant has been shutdown. For the first couple of hours, the thermal system of the power plant is still heated, so it can be restarted quickly. The longer the power plant has been switched of, the longer it takes to reactivate it. However, the re-activation time does not follow a purely linear curve, since auxiliary systems can be switched off after a certain time to save costs. The piecewise linear function is encapsulated by a decision expression startupSteps that returns the steps required to start the power plant depending on the duration it has been stopped so far.

```
// needs separate vectors since tuples cannot contain sets in excel data (12.2)
float slopes[ControllablePlants][1.. maxBps+1] = ...;
float breakpoints[ControllablePlants][1.. maxBps] = ...;

// two times the same breakpoint (32, 32) indicate a step
// rather than a slope at slope[i+1] with i being the index
// of the first occurrence of breakpoint[i]
pwlFunction startUpFunction[p in ControllablePlants] = piecewise(i in 1..PowerPlant[p].numBPs)
{
   slopes[p][i]->breakpoints[p][i]; slopes[p][PowerPlant[p].numBPs+1]
} (0, PowerPlant[p].fAtZero);

// start up time depending on stop time
dexpr float startUpSteps[p in ControllablePlants][t in TIMERANGE] = startUpFunction[p](consStop[p][t]);
```

A number of decision expressions are used to facilitate the formulation of constraints. These expressions exists for running plants (shown here) and stopped plants. The expression consRunSum sums the timesteps in which the power plant has been running. Whether the plant has just been shut off is indicated by fallingEdge. The expression consRun uses subtractRun to calculate the time a power plant has been consecutively running.

```
// decision expressions for consecutive running time
dexpr int consRunSum[p in ControllablePlants][t in TIMERANGE] =
  sum(t2 in 0..t) powerPlantRunning[p][t2] + PowerPlant[p].consRunningTimeInit;
dexpr int fallingEdge[p in ControllablePlants][t in TIMERANGE] =
  powerPlantRunning[p][t] == 1 && powerPlantRunning[p][t+1] == 0;
dexpr int subtractRun[p in ControllablePlants][t in TIMERANGE] =
  maxl(0, max(t2 in 0..t-1) fallingEdge[p][t2] * consRunSum[p][t2]);
dexpr int consRun[p in ControllablePlants][t in TIMERANGE] = consRunSum[p][t] - subtractRun[p][t];
```

Now, two new decision variables can be introduced. The startSignals variable indicates whether a power plant should start up. The variable countDown counts the number of time steps until the power plant is operational. The expression isStarting indicates based on the count down, whether a power plant is currently starting.

```
dvar boolean startSignals[p in ControllablePlants][t in CONSIDERED_RANGE] in 0 .. 1;
dvar int countDown[p in ControllablePlants][t in CONSIDERED_RANGE] in 0..inf;
dexpr int isStarting[p in ControllablePlants][t in CONSIDERED_RANGE] = countDown[p][t] <= (inf-1);
```

Constraints are used to enforce the minimal stop times and the minimal run times. If a power plant has been switched off (or on) as indicated by the change in powerPlantRunning, the consecutive run times (or stop times) have to adhere to the minimal times indicated.

```
// minimal running/standing time constraint
forall (t in 1..lastSimStep) {
  minStopTimeConstraint:
    (powerPlantRunning[p][t] == 1 && powerPlantRunning[p][t-1] == 0) =>
      consStop[p][t-1] - PowerPlant[p].minOffTime >= 0;
  minRunTimeConstraint:
    (powerPlantRunning[p][t] == 0 && powerPlantRunning[p][t-1] == 1) =>
      consRun[p][t-1] - PowerPlant[p].minOnTime >= 0;
}
```

Finally, the count down and the relationship between the start signals and the power plant status has to be established. The first two constraints enforce coherent values for the isStarting expression by limiting its values according to power plant state and count down values. The constraint

initCountDownAfterStartSignal sets the count down if a start signal has been issued. Each value in the count down is set to infinity if the power plant is not starting by the fourth constraint. Finally, decreaseConstraint reduces the count down by one over the entire course of the startup phase.

```
// count down constraints
forall(t in TIMERANGE) {
  resetIsStartingConstraint: // when plant is running, reset state
    (powerPlantRunning[p][t+1] == 1) =>
      ((countDown[p][t+1] == 0) && (isStarting[p][t+2] == 0)) || (powerPlantRunning[p][t] == 1);
  justifyIsStartingState : // there has to be a reason for isStarting to be set true
    (isStarting[p][t+1] == 1) => ((isStarting[p][t] == 1) || (startSignals[p][t+1] == 1));
  initCountDownAfterStartSignal : // after a start signal is placed, countDown has to be adapted
    (startSignals[p][t] == 1) => (countDown[p][t+1] == startUpSteps[p][t] && isStarting[p][t] == 1);
  defaultCountdown: // if no start-progress set cntDown
    (isStarting[p][t] == 0) => countDown[p][t] == inf;
  decreaseConstraint:
    (isStarting[p][t] == 1 && countDown[p][t] >= 1) => (countDown[p][t+1] == countDown[p][t] - 1);
}
```

The extension of the constraint optimisation model for power plants that can be switched off and for standby and startup times introduce new constraints, decision variables, and constraints. Each new decision variable introduces a new degree of freedom into the system and has to be allocated by the solver accordingly.

Hierarchy Formation

The formation of hierarchies of AVPPs and power plants is driven by two types of constraints, as described in Chapter 4. The two main constraints constituting the first type are responsible for the introduction and dissolution of hierarchy levels. They constrain the time required to solve the scheduling problem. The second type of constraints is responsible for the anticlustering within a hierarchy level. They constrain the composition of the AVPPs and enforce their similarity. These are the same constraints that are used in the non-hierarchical partitioning with SPADA.

The two types of constraint are observed at runtime as described in Chapter 6. The constraint that causes the introduction of a new hierarchy level, e.g., evaluates the scheduling runtime. As this data is ascertained at runtime and is influenced by a number of factors, it is not, however, available to a constraint solving algorithm that has to create a new hierarchical structure. Therefore, alternative constraints have to be used in the specification of the problem. This is similar to the scheduling problem where the actual output is unknown at solution time and instead predictions are used. For hierarchy formation, we can use knowledge about the complexity of the scheduling problem and the controllability—i.e., the amount of power they can produce and how fast the output can change—of the power plants to estimate the scheduling times that will be required and the potential solution quality of a partitioning.

9.2 Constraint Relationships for the Specification of Optimisation Criteria

Constraint relationships (i.e., soft constraints) allow the expression of preferences over constraints and thus of optimisation criteria (cf. Chapter 5). This ability can be used in a number of ways in the definition of constraint satisfaction problems for adaptive systems. The constraint models for power plant scheduling in the previous section contained one optimisation criterion, namely the minimisation of violations. In actual scheduling problems, however, many more aspects play an important role. Power plants should, e.g., not be constantly shut down only to be started up again shortly after. Similarly, the possibilities provided by the physically possible rate of change should not be utilised constantly in thermal power plants as heating up the system and cooling it down again wastes enormous amounts of resources. Indeed, as such restrictions depend very much on the concrete power plant, a certain heterogeneity in the models is required as discussed in the next chapter.

To include soft constraints in the constraint problems shown above, they are extended with additional constraints. The scheduling problem is used as an illustration here since the inertial behaviour of thermal power plants and the economic “sweet spot” at which power plants are operated optimally have not been taken into consideration yet. These factors, however, play an important role in the economic

optimisation of power plant operation. To include these factors in the constraint optimisation problem, they can either become part of the objective function—e.g., by assigning costs to changes in output or by providing a cost function that assigns production targets not in the economic optimum a higher cost—but these measures make the optimisation problem harder since the solver now has to deal with a pareto-function and with non-linear cost functions.

Instead, additional constraints can be defined and soft constraints can be used to express optimisation criteria. While the economically optimal production of a power plant is generator-specific and will be covered in Chapter 10, inertia can be dealt with globally. The inertia of thermal power plants is due to the fact that changes in output require cooling or heating the thermal system. This is either associated with using large quantities of cooling water or by using additional fuel to heat the system up. Both changes are relatively expensive. It is therefore advantageous to operate a thermal power plant at a relatively steady output. Especially, *thrashing*, i.e., increasing the production—heating the system up—followed by a decrease in production—cooling the system down—and a subsequent increase in production should be avoided, even though they are physically possible and not prohibited by the constraints shown above. In essence, the number of *load changes* within a certain period of time has to be limited. Thus, a MaxSPAN constraint (cf. Section 6.2) can be used. The constraints to limit the rate of change are rather simple:

```
// Limit the rate of change to 15% to avoid oscillating power production (soft constraint)
limitRateOfChangeConstraintStrict :
  (powerPlantRunning[p][t] == true) =>
    abs(energyProduction[p][t] - energyProduction[p][t+1]) <= energyProduction[p][t] * 0.15;
// Limit the rate of change to 25% to avoid oscillating plants (soft constraint)
limitRateOfChangeConstraintRelaxed :
  (powerPlantRunning[p][t] == true) =>
    abs(energyProduction[p][t] - energyProduction[p][t+1]) <= energyProduction[p][t] * 0.25;
```

Avoiding thrashing, however, is a little more complicated. In essence, we need to evaluate the number of load changes in a given time horizon. A load change occurs whenever the production was increased first, then decreased, or vice versa. To easily evaluate if a load change occurred, a vector of booleans is most handy since we can simply sum the number of trues in a section of the vector and get the number of load changes that occurred. The corresponding constraint, limiting the number of changes in a time window of 12 time steps (the value for r) to a maximum of 4 is shown below.

```
// Limit changes in subsequent iterations (soft constraint)
limitRateOfChangeSubsequent :
  (powerPlantRunning[p][t] == true) =>
    sum (r in t-r..LAST_SIMULATION_STEP-1) loadChange[p][r] < 4;
```

Constructing the loadChange[p][t] vector is a little more complicated. We first define two decision expressions prodDiffNeg and prodDiffPos that contain values of 1 whenever there was a reduction in output or an increase in output respectively. These expressions are then combined in an aggregated expression prodDiffAgg that contains a -1 for a reduction in output, a 1 for an increase and a 0 if the production stayed the same. This expression can in turn be used to define the final loadChange vector:

```
// The difference in production. 1 if negative, 0 if positive
dexpr float prodDiffNeg[p in ControllablePlants][t in 0..card(TIMERANGE)-2] =
  energyProduction[p][t] - energyProduction[p][t+1] <= 0.001;
// The difference in production. 1 if positive, 0 if negative
dexpr float prodDiffPos[p in ControllablePlants][t in 0..card(TIMERANGE)-2] =
  energyProduction[p][t] - energyProduction[p][t+1] >= 0.001;
// The aggregated difference in production
dexpr int prodDiffAgg[p in ControllablePlants][t in 0..card(TIMERANGE)-2] =
  prodDiffPos[p][t] - prodDiffNeg[p][t];
// A vector indicating load changes
dexpr int loadChange[p in ControllablePlants][t in 1..card(TIMERANGE)-1] =
  loadChange[p][t] = prodDiffAgg[p][t-1] + prodDiffAgg[p][t] == 0
                     && prodDiffAgg[p][t-1] <> 0 && prodDiffAgg[p][t] <> 0;
```

These constraints are identified as soft constraints by defining constraint relationships in which they contribute.

```
limitLoadChanges          >> limitRateOfChangeConstraintStrict
limitLoadChanges          >> limitRateOfChangeConstraintRelaxed
limitRateOfChangeConstraintRelaxed >> limitRateOfChangeConstraintStrict
```

The weighting algorithm shown in Section 5.4 is then applied to calculate their respective weights. The constraints are changed and a new term penalty is added. The constraint is either fulfilled or the penalty vector contains the penalty at the index assigned to the constraint. The calculation of the weights uses the *transitive predecessor dominance* criterion.

```
forall ( t in 0..card(TIMERANGE)-2 ) {
// Limit the rate of change to 15% to avoid oscillating power production (soft constraint)
limitRateOfChangeConstraintStrict :
  ((powerPlantRunning[p][t] == true) =>
    abs(energyProduction[p][t] - energyProduction[p][t+1]) <= energyProduction[p][t] * 0.15)
    || (penalty[2][t] == 1);
// Limit the rate of change to 25% to avoid oscillating plants (soft constraint)
limitRateOfChangeConstraintRelaxed :
  ((powerPlantRunning[p][t] == true) =>
    abs(energyProduction[p][t] - energyProduction[p][t+1]) <= energyProduction[p][t] * 0.25)
    || (penalty[5][t] == 2);
}
forall ( t in 1..card(TIMERANGE)-1 ) {
// Limit changes in subsequent iterations (soft constraint)
limitLoadChanges :
  ((powerPlantRunning[p][t] == true) =>
    sum (r in t-r..LAST_SIMULATION_STEP-1) loadChange[p][r] < 4) || (penalty[6][t] == 5);
}
```

While the previous models used an objective function that minimised the difference between demand and production, the usage of weights makes it necessary to change the minimisation goal. As introduced in Section 5.4, the weights assigned to the constraints can be used to define an objective function that minimises the sum of the weights of violated constraints. If an objective function is already present, however, this approach introduces a pareto optimisation again, an added complexity that we aimed to avoid. In addition, the two variables in the pareto optimisation function have no direct relation to each other—the difference between the production and the demand and the weight of the violated constraints are two completely different things. This makes it difficult to select a point on the pareto-front. Since the difference between demand and production—the violation—is about a factor of 1000 greater than the maximum penalty in the problems regarded here (see below for a brief evaluation), we can match both factors by dividing the violation by this factor.

```
range PENALTYRANGE = 1..7;
dvar int+ penalty[PENALTYRANGE][TIMERANGE];

minimize
//  sum ( t in TIMERANGE ) abs(violation[t]);
//  sum (i in PENALTYRANGE, t in TIMERANGE) penalty[i][t];
  (sum ( t in TIMERANGE ) abs(violation[t]) * 0.001) +
        (sum (i in PENALTYRANGE, t in TIMERANGE) penalty[i][t]);
```

Instead, constraints could again be used to limit the violation. Constraint relationships are a way to express the requirement to have as little deviation from the optimum as possible. This can be done by using escalating constraint relationships. The corresponding constraints are shown below, already weighted according to a transitive predecessor dominance semantic. The constraint relationship graphs for both the limitation of the rate of change and the limitation of the violation are shown in Figure 9.4.

```
// Demand must always be satisfied (violation[t] allows deviation from full satisfaction)
satisfyDemandConstraint:
  totalProduction[t] + violation[t] == energyConsumption[t];
// Enforces that the violation is less than 0.5% of total production (soft constraint)
noViolations:
  (abs(violation[t]) <= totalProduction[t] * 0.005) || (penalty[1][t] == 1);
// Limits the violations to 1% of total production (soft constraint)
```

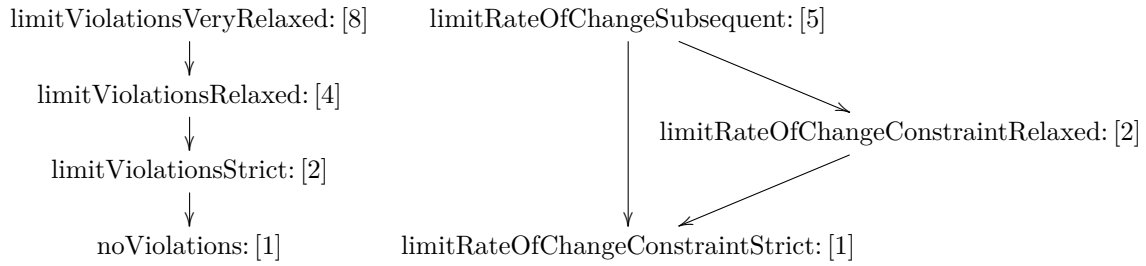


Figure 9.4: The constraint relationship graphs for the soft constraints limiting the the violation (left) and the rate of change (right) in the power plant scheduling problem with corresponding weights in a transitive predecessor dominance relationship.

```
limitViolationsStrict :
  (abs(violation[t]) <= totalProduction[t] * 0.01) || (penalty[2][t] == 2);
// Limits the violations to 3% of total production (soft constraint)
limitViolationsRelaxed:
  (abs(violation[t]) <= totalProduction[t] * 0.03) || (penalty[3][t] == 4);
// Limits the violations to 5% of total production (soft constraint)
limitViolationsVeryRelaxed:
  (abs(violation[t]) <= totalProduction[t] * 0.05) || (penalty[4][t] == 8);
}
```

Comparison of Solution Approaches with Pareto-Optimisation and Pure Penalty Minimisation

Of course, this last version of the problem statement might give the optimiser more freedom than he would have had if he tried to minimise the violation. In a pure minimisation, the violation might have been 0 while the solution for the problem with soft constraints might be within the 0.5% allowed by the noViolations constraint. Such leeway depends very much on the problem and especially in small AVPPs, a violation of 0.5% might be admissible. In general, using constraint relationships in this way requires tweaking the parameters of the constraints. If a pareto-optimisation function is used, both terms need to be balanced so their values are in a similar domain. Otherwise, one can dominate the other and the dominated variable does not contribute to the solution. If the violation is much greater than the penalty, the values for the penalties will not play a role in the minimisation. As discussed above, a factor of 1000 is used in the experiments to scale down the violation to the same level as the penalty.

The performance of the solution process depends critically on the algorithm used. A continuous version that uses float values for all decision variables can be solved optimally by the CPLEX solver within 12 minutes for a time horizon of 8 time steps. Increasing the time horizon to 12 time steps, however, increases complexity enough to make the solution process run for more than 80 hours without finding a solution to the problem. When mathematical programming is used—the CP solver, basically a branch-and-bound algorithm—solution times for optimal solutions rise to much higher values than the 12 minutes in the continuous problem. However, incorporating soft constraints and penalties reduces the time until a viable solution is found. As the branch-and-bound algorithm is an anytime algorithm, processing can be stopped after a certain time or when a solution with an acceptable objective is found. Table 9.1 shows different solutions for individual solution runs of the different variants of the model.

The results demonstrate that the discrete model with an optimisation for violations only is not suitable for the scheduling problem. In general, using a continuous model yields much better results for power plant scheduling since faster algorithms can be used (linear programming, especially) and the solver benefits greatly from the ability to assign fractions to the decision variables. In addition, the problem is especially difficult due to the required power plant startup. Once an initial start-up phase is completed, the model that optimises only for the penalty performs reasonably well. However, good results only become available after 5 time steps instead of 3 in the continuous version. Runtimes exceeding those used in this evaluation might yield better results, however experience shows that this is not necessarily the case. Especially when minimising penalties, good solutions are usually found quickly and do not improve later on. The solution in Table 9.1, e.g., was found after 191 seconds and was not improved within 12 hours.

	Continuous	Discrete, optimised for violation only	Discrete, optimised for penalty only	Discrete, pareto-optimisation
Timestep	Violation	Violation	Violation	Violation
0	21185.54	21186	21185.54	21185.54
1	20545.78	20546	20545.78	20545.78
2	18937.98	18936	18937.98	18937.98
3	0	16665	16665.27	16665.27
4	0	15004	15004.14	15004.14
5	0	9459.1	205.10	6934.10
6	0	8478.3	200.34	5913.34
7	7.28	7272.8	196.75	5134.75
Violation	60667.301	117549	92940.90	110320.90
Penalty	N/A	121	75	120
Solution time	12:27:20	33:35:95	36:35:57	35:54:11

Table 9.1: Comparison of solution runs for different variants of the power plant scheduling model. Schedules were created for a total of 8 time steps with controllable power plants in their off-state at time step 0. The continuous model does not take soft constraints into account, the other models differ in the objective function as indicated in the table caption. Scheduling was performed for 160 controllable power plants. Violation indicates the difference between demand and production. Violations in the first three time steps are due to the start-up times of the power plants. Solution runs of the discrete models were stopped after 10000 seconds of used CPU time or 500 found solutions.

The discrete model that minimises only the penalty yields a result that is close to the optimum within the boundary defined by the noViolation constraint for the three final time steps. In summary, these results indicate that it is feasible to combine both approaches: find an initial solution with the continuous model and use the discrete one to optimise for the preferences of the power plants. Implementation and evaluation of such an approach is left as future work, however.

Violation of Time-indexed Constraints

The current formulation has a very specific semantic that should be specifically regarded. As the constraints are evaluated for each of the time steps t **in** TIMERANGE and a violation causes an assignment of the penalty for each time step, penalties are accumulated over time. In principle, the optimiser solves a CSP for each time step but constraint violations are not considered in each individual step but aggregated over all time steps. While this is intuitively the desired behaviour, it introduces problems when combining such constraints with ones that are not time-indexed and thus have no summed up penalty. As the summation increases the penalty accordingly, the constraints become "more important" than others as their aggregated weight can far surpass the weight of a constraint that is not summed up. In the example shown here, all constraints that are in a relationship with each other are time-indexed and therefore, consistency is at least adhered to internally. Nevertheless, such a formulation destroys the partial order of constraints that is introduced with constraint relationships and should be used only with care.

9.3 Integration of Trust and Prognoses of Future Behaviour

The scheduling problem is an excellent example of an optimisation task that has to deal with imperfect information. In practice, schedules for power plants must be created based on a number of uncertain predictions and assumptions *at runtime*. While the models shown above where meant for offline development and used the power consumption from recorded data and used a pre-defined solar radiation and wind strength to calculate the production from intermittent power generators, an online power plant scheduler has to employ predictions from consumers and producers to determine these factors. Since the predictions made from consumers and producers are inherently uncertain, different techniques can be employed to correct the uncertainty contained in the data and create expectancy values that make using uncertain data more robust and allows the calculation of better solutions.

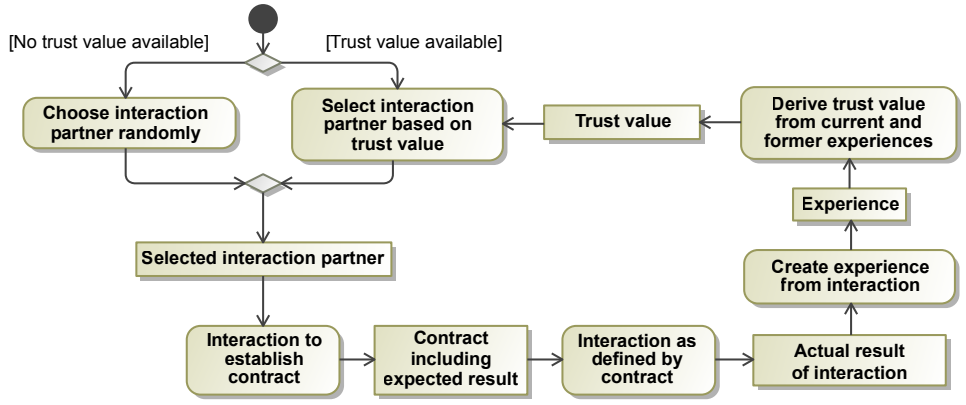


Figure 9.5: The life-cycle of trust values. A *contract* defines the expectancy that both parties have for the result. An interaction takes place and both parties generate an experience that captures the difference of the contract to the result. This information is used to generate a trust value which can in turn help in selecting interaction partners in the future.

Using Trust Values to form Expectations

The key element in the incorporation of uncertain value in online optimisation is the use of *trust values*. As outlined briefly in Section 2.6, trust values can be used to increase predictability and at the same time reduce the risk associated with using predictions. In autonomous power management systems and other automated decision systems, there is a risk that misdeterminations or unforeseen emergent behaviour can bring the system into jeopardy. This risk is mainly associated with the quality of predictions and scheduling decisions based on them. Therefore, the objective of an autonomous power management system must be to increase predictability and minimise this risk.

Trust values are created through observation of past behaviour and subsequent analysis of these experiences. The basis of a trust value is a contract two or more parties commit to (Ramchurn et al., 2004a) that defines an *interaction* (possibly composed of several distinct steps) as well as its *expected result*. In case of predictions of intermittent power generators, the contract is implicit, stating the expectancy of the requesting agent—i.e., the agent that will perform the scheduling—that the generator will provide a prediction that is as close as possible to the generated power. This generated power is the *actual result* of the interaction that can be compared to what was expected according to the contract, thus yielding an *experience* for each party. Experiences can in turn be used to generate *trust values* which inform future interactions as depicted in Figure 9.5. The derivation of trust values is determined by a trust model. In many circumstances, trust is measured as a numerical value, often ranging between 0 and 1. The trust model is basically an algorithm whose input, the experiences, are transformed into this value. Of course, such a value has to be semantically sound, a virtue that depends on the concrete trust model used.

The use of trust values is based on one important assumption: past behaviour must be an indication of future behaviour. If the experiences an agent made with another one show erratic behaviour, a trust value is not useful. Only a systematic deviation from a prediction, a pattern of behaviour yields a trust value that can be used to form an expectation about the result of future interactions.

To gauge the quality of predictions of power plants, it is useful to use a model that produces values in the range $[-1, 1]$ to be able to deal with predictions that are lower and higher than the actual production. A trust model[2] is defined as $\mathcal{M} : \mathcal{E} \times ... \times \mathcal{E} \to \mathcal{T}$ and evaluates a number of experiences whose domain is $\mathcal{E}$ to calculate a value in $\mathcal{T}$. Each experience that it is given as its input is individually evaluated with a rating function $\mathcal{R} : \mathcal{E} \to \mathcal{T}$. The resulting $\pi \in \mathcal{T}$ are then combined as defined by $\mathcal{M}$. In the case of power predictions, the metric is simply an arithmetic mean of the individual ratings. The ratings themselves depend on the deviation from the contract and maximum production of the power generator. An experience $E_t \in \mathcal{E}$ for a point in time t consists of a value stipulated in the contract c_t and an actual value r_t. An additional value $k \in \mathbb{R}$ equals the maximum possible—or, if not available, observed—deviation from a contract and thus normalizes the result to a value in $[-1, 1]$:

[2]This definition is a simplification of the model used in (Anders et al., 2013a).

$$\mathcal{R}(E_t) = \frac{r_t - c_t}{k} \tag{9.1}$$

The ratings can easily be combined and divided by their number to yield a simple trust value:

$$\mathcal{M}(E_{t_1}, ..., E_{t_m}) = \frac{\sum_{h=1}^{m} \mathcal{R}(E_{t_h})}{m} \tag{9.2}$$

Example—Trust Values of a Solar Power Plant

To illustrate the use of trust values to correct deviations from the predictions of power plants, we assume a solar power plant with a rather unsophisticated prediction algorithm. Figure 9.6 shows the predictions and the actual output of the power plant. The deviation is rather substantial. If power plant scheduling would have used the predictions, the controllable plants would not have produced enough power and the system could have become unstable. At the very least, the controllable plants would have had to be rescheduled.

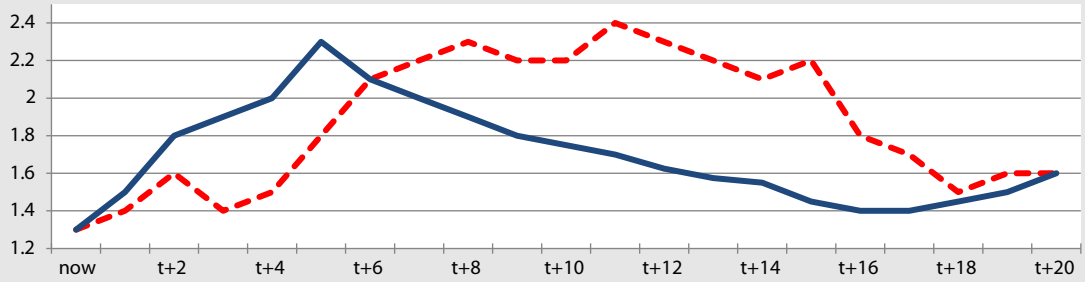


Figure 9.6: The predictions (red dashed line) and the actual output (blue solid line) of a solar power plant with a maximum output of 2.5kW over 20 time steps

The figure also shows that the deviation of the power plant changes. In the first six time steps, the prediction is less than the produced power, in the 14 time steps after it is substantially more. Therefore, a sliding time window is used to only regard recent behaviour. A better approach to capture such a behaviour are trust-based scenarios (see below). The sliding time window has a size of 6, meaning that the 6 most recent experiences are used as input to $\mathcal{M}$. The metric then calculates a trust value based on the ratings of the experiences which is in turn used to create an expected value by applying the formula $e_t = c_t + (c_t * \mathcal{M}(E_{t-1}, ..., E_{t-6}))$. For time step $t + 16$, e.g., the sliding time window yields a trust value of -0.095. The predicted output is 2.3kW, the expected output is 2.0815kW respectively. The expected output is contrasted with the entire series of predictions and actual outputs in Figure 9.7.

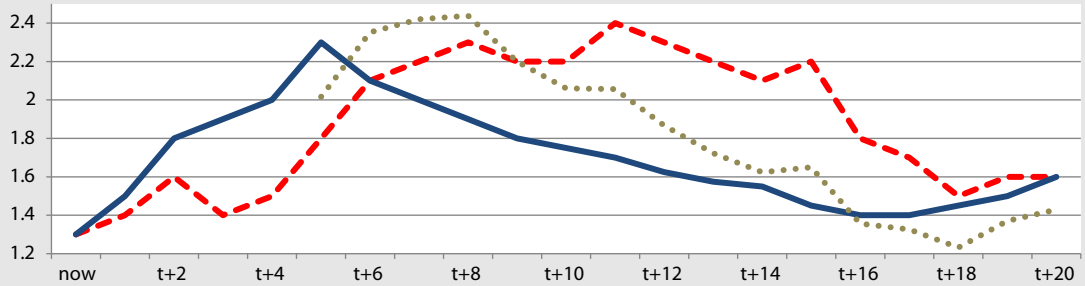


Figure 9.7: The expected production (green dotted line) compared to the predictions (red dashed line) and the actual output (blue solid line) of a solar power plant with a maximum output of 2.5kW over 20 time steps

The example shows that while the expected value based on trust values improves the predictions for most time steps, areas in which the behaviour changes are hard to capture. Again, trust-based scenarios can be applied in such cases. Another possibility is to use confidence values (Kiefhaber et al., 2012) that calculate how well a trust value depicts the behaviour of a process based on factors such as the standard deviation of measurements.

In general, creating an expected value e_t from a contractual value c_t based on past experiences $\mathcal{E}$ and a time window m can be expressed as

$$e_t = c_t + (c_t * \mathcal{M}(E_{t-1}, ..., E_{t-m})) \tag{9.3}$$

with E_{t-i}, $1 \leq i \leq m$. Such an expected value is created in a pre-processing step online in the application. The constraint satisfaction problem has only be altered insofar as instead of using predictions directly, it now uses expectations. A change in the formulation of the problem is not necessary. As shown in Anders et al. (2013a) and in the evaluation below, the use of expected values already enhances the scheduling accuracy significantly.

Trust-based Scenarios for Robust Optimisation

As mentioned above, trust values can be used as measures of uncertainty to make evidence-based assumptions about an agent's future behaviour. However, a trust value as an aggregated value is not sufficient to capture the complexity of the stochastic process underlying the agent's actual behaviour. It can thus only give a very imperfect picture of the interaction partner. In addition, it does not allow to differentiate different behaviours that have occurred in the past. As the experiences are aggregated, even simple conditional behaviours are not detected. If an agent would always behave like the solar power plant in the example, i.e., underestimating the actual output first and subsequently overestimating it, such information could be used to formulate more accurate expectations about the future behaviour if an underestimation is observed.

Trust-based scenarios are a tool that allows leveraging such information at runtime. Scenarios are used in operations research to mirror a system's underlying stochastic process (Hochreiter and Pflug, 2007). They express different possible developments of a system. As each scenario has a certain probability of occurrence, an agent can, e.g., choose the most likely scenario and optimize for the future the scenario predicts. However, current approaches—especially in the domain of power management—usually use predetermined scenarios and probabilities (Zhang et al., 2011). In an open self-organizing system, it is not possible to determine possible scenarios beforehand. As self-interested agents come and go and behavioural patterns change, no a priori assumptions about the individual agent's behaviour can be made. It is thus essential that scenarios and their probabilities are determined at runtime with up-to-date data.

Trust values can be used as the basis for dynamically calculating trust-based scenarios and their probabilities at runtime. They provide a model for sources of uncertainty and allow agents to form expectations that differ from the predictions, improving them and allowing more robust decisions. Different scenarios and their respective probabilities are captured in a scenario tree. In comparison to pure trust values, they regard time-dependent behaviour and provide the opportunity to select different paths based on past experiences. In essence, sequences of experiences are classified according to their rating. This classification is repeated for a large number of sequences. The resulting classification is transformed into a tree that provides transition probabilities between the different classes. A path in the tree equals a scenario. The following example shows the creation of scenario tree based on the prediction behaviour of a power plant as shown above. A formal description of scenario trees can be found in (Anders et al., 2013a).

Example—Trust-based Scenarios for a Solar Power Plant

If we consider the solar power plant in the example above and assume the predictions and the actual output as a blue print for the power plants behaviour, it seems to have two distinct characteristics: it notoriously underestimates its output when the actual output is relatively high and it overestimates its output when the actual output is relatively low. Put differently, if the rating was positive—i.e., the power plant underestimated its actual output—it is bound to stay positive; if the rating was negative—i.e., the power plant overestimated its actual output—it is bound to stay negative. An interesting behaviour can be observed at those points where the behaviour changes (phase transitions in the parlance of Chapter 8).

In the table below, four behaviours are distinguished. In time steps t_2 to t_4, the power plant underestimates its output. In t_5 to t_6, this behaviour changes towards the one observed in t_{12} to t_{14}

where the actual output is overestimated. In steps t_{18} to t_{20}, the prediction fits the actual output well. The trust value for the respective three successive experiences is also shown in the table.

	t_2	t_3	t_4	t_5	t_6	t_7	t_{12}	t_{13}	t_{14}	t_{18}	t_{19}	t_{20}
Predicted Output	1.6	1.4	1.5	1.8	2.1	2.2	2.3	2.2	2.1	1.5	1.6	1.6
Actual Output	1.8	1.9	2.0	2.3	2.1	2.0	1.625	1.575	1.55	1.45	1.5	1.6
Rating	0.08	0.2	0.2	0.2	0.0	−0.08	−0.27	−0.25	−0.22	−0.02	−0.04	0.00
Trust vale ($m = 3$)		0.093			0.067			−0.173			−0.02	

To construct a scenario tree from these behaviours, they have to be classified. The ratings are used as classifiers of each experience. Experiences belonging together are connected to each other. Figure 9.8 shows the classifications of the behaviours above.

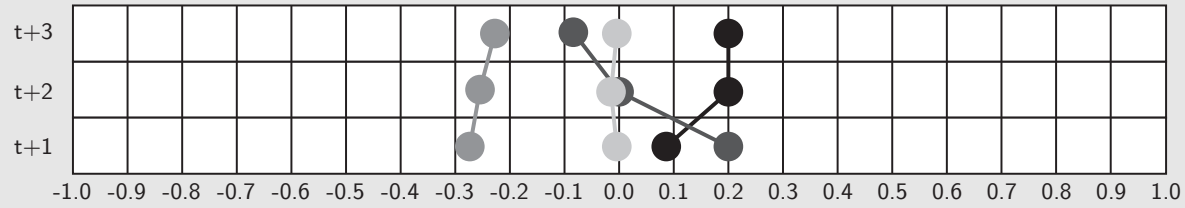


Figure 9.8: The behaviours from the table above classified according to their rating. The x-axis shows the valuations of the classes and the y-axis the time steps, relative to the current point in time.

To construct a valid scenario tree, many more experiences are usually used. The more information is available, the better will the scenario tree be able to depict the actual behaviour of the system. However, as a system can change its behaviour so that old experiences are no longer valid, a trade-off has to be made between adaptability to changing behaviour and prediction quality. Figure 9.9 shows the classification scheme extended with additional observations that will be used to construct the scenario tree.

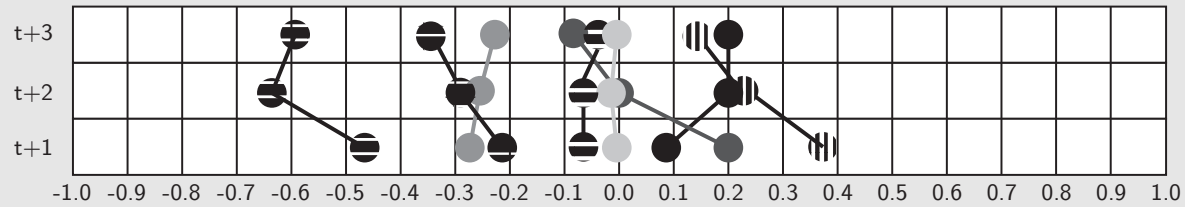


Figure 9.9: An extended set of behaviours classified according to the same scheme as before.

Now, the scenario tree can be constructed. For this purpose, the root of the tree is introduced first. Starting from the root, a node is created for each of the classes that contains an experience. The node is annotated with the number of experiences in each class. The result is shown in Figure 9.10.

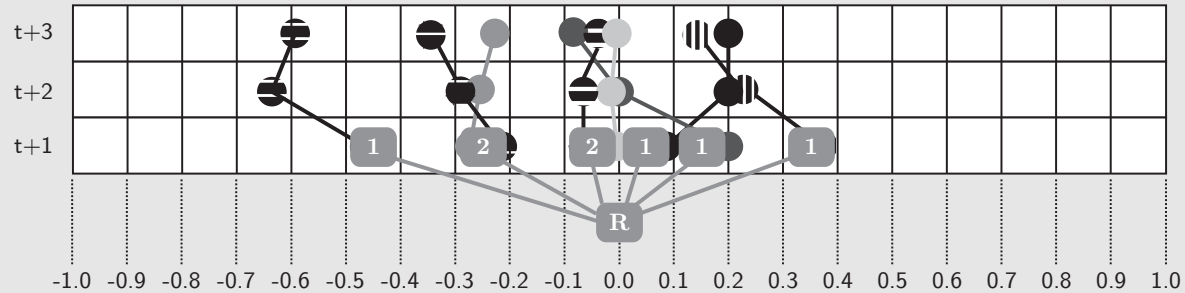


Figure 9.10: The first step in the construction of a scenario tree. A root node is introduced and new nodes are added for all classes that contain at least one experience. The nodes are annotated with the number of experiences in the class.

In the second step, the development of the behaviour is traced. Each of the connections of the experiences is followed and a new node is introduced for each unique transition. This becomes evident in the three highlighted nodes in Figure 9.11: the class $(-0.1, 0.0]$ in $t + 2$ contains two nodes since there were two unique transitions into the class. However, the experiences that are represented by the node both transitioned to the same node in $t + 3$ and thus, a single node for the class is created in this time step and annotated accordingly.

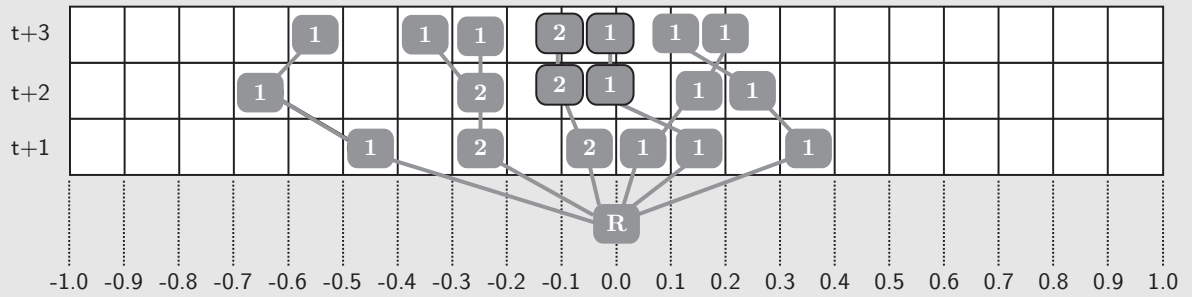


Figure 9.11: The second step in the construction of a scenario tree. Transitions in the original tree are followed and new nodes are introduced for each new class reached by the transitions. The nodes are labelled with the number of transitions that lead from one node in the prior time step into the class.

The tree can now be transformed to depict the classes and the transition probabilities according to the number of occurrences in the classification. In Figure 9.12, the transition probabilities have been multiplied to yield the probability of each individual scenario, defined by a unique path from the root to a leaf node.

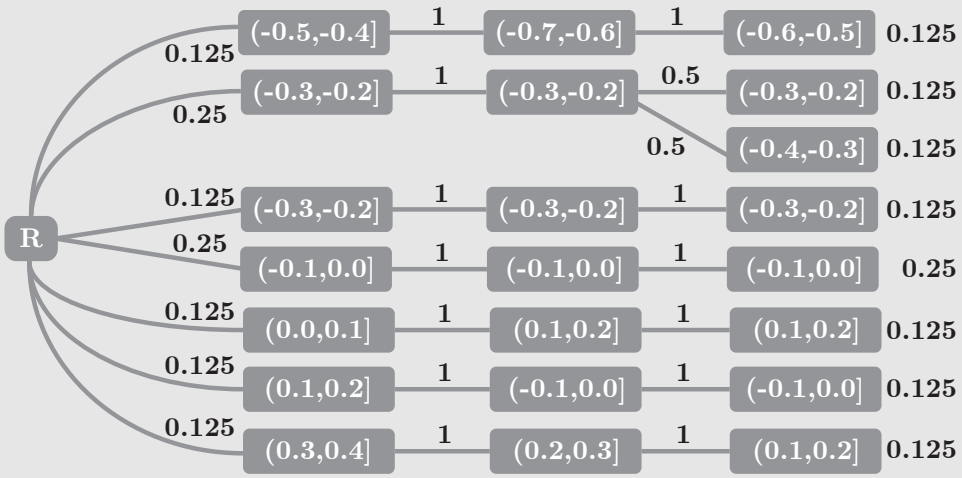


Figure 9.12: The final scenario tree. Nodes represent classes and transitions are labelled with their probabilities. The probabilities of one scenario are the product of the probabilities of all transitions involved.

The information in the scenario tree can now be used as the basis of an optimisation. For this purpose, one or more scenarios are selected from the scenario tree. Selection can be based on probability or other metrics, e.g., to assess the cost or the risk of a scenario.

Scenario trees allow taking several possible developments of a system—the scenarios—into account (see, e.g., Pappala and Erlich, 2008). They serve as input for solving optimization problems under uncertainty such as stochastic programming (Sahinidis, 2004). These techniques are applied, e.g., in the domain of power management systems (see, e.g., Densing, 2007; Bouffard and Galiana, 2008; Zhang et al., 2011). In literature, scenarios are generated, e.g., by solving an optimization problem on the basis of a system model, gathered historical data, and expert knowledge (Hochreiter and Pflug, 2007) that approximates a continuous probability distribution, similar to kernel density estimation. Expert

knowledge can be used to identify relevant data and influence scenario generation.usually predefined (see, e.g., Densing, 2007; Bouffard and Galiana, 2008). Due to the computational complexity, scenarios are often determined off-line (Zhang et al., 2011). In comparison, trust-based scenarios can be generated online with very little computational effort and be adapted whenever new knowledge becomes available.

Incorporating scenarios into the constraint satisfaction problems discussed in Section 9.1 and Section 9.2 requires replacing data used for predictions by data gathered from scenarios. It is either possible to optimise for a single scenario—making very little adjustment to the models necessary—or optimising for several scenarios at once which requires a little more work. The latter case can be demonstrated in a slightly different formulation of the scheduling problem in which the expected production price is minimised. Differences in production and demand are incorporated into the price so they are being dealt with. Scenarios are created for the *residual load*—the load that has to be fulfilled by controllable power plants. It is calculated from the predictions of consumers and of intermittent power sources, all of which are inflicted by uncertainty. Intuitively, the scenario trees from each individual source of uncertainty are combined into one scenario tree for the residual load. The tree has considerably more classes than in the example above (more than 180) to be able to capture small differences in the residual load which is a value that can be in the megawatt range.

Encoding a scenario tree in a way that makes it usable in a constraint satisfaction problem requires some additional constants. First, a CLASSRANGE is defined to allow accessing the lower and upper bounds of each class. A corresponding tuple includes a time step and these bounds. It thus represents one of the classes in Figure 9.12. A set of classIds is used to address each of the classes. Finally, the residualLoad is an array indexed over the classIds in which each field contains one of the classes.

```
// 0 represents the scenario class's minimum, 1 its maximum value.
range CLASSRANGE = 0 .. 1;
// A class used to model different scenarios for the residual load
tuple class {
  int timeStep;
  float minValue;
  float maxValue;
}
// A list of unique ids, each identifying a specific class
{string} classIds = ...;
// An array that holds all states for the residual load that occur in scenarios
class residualLoad[classIds] = ...;
```

The total number of scenarios is used to define a corresponding SCENARIORANGE. Each scenario within the range has a probability. The conncetion between the classes indexed over their ID, the index of the scenario in the range and the timestep is established through classMapId.

```
// The number of scenarios for the residual load
int nbOfScenarios = ...;
// Range of scenarios for the residual load
range SCENARIORANGE = 1..nbOfScenarios;
// The probability of a specific scenario
float scenarioProbability[SCENARIORANGE] = ...;
// A map whose cells can be adressed by a tuple (pathId,timeStep).
// The cell contains the corresponding stateId for the residual load.
string classIdMap[SCENARIORANGE][TIMERANGE] = ...;
```

Now, the decision variables are defined. Each class has an energy production for its upper and lower value. Likewise, each scenario has an energy production. As the production is determined by the sum of the production of the plants, the variables are indexed accordingly. Each scenario also causes a violation, i.e., a difference between the total production and the residual load.

```
// Energy production for a specific class
dvar float energyProductionClass[classIds][CLASSRANGE][ControllablePlants];
// The energy adjustment to satisfy a specific scenario (corresponds to a part of y_w).
dvar float energyProductionScen[SCENARIORANGE][CLASSRANGE][ControllablePlants][TIMERANGE];
// The energy missing or superfluous to meet demand in each time step dependent on a specific scenario
dvar float scenarioViolation[SCENARIORANGE][CLASSRANGE][TIMERANGE];
```

To determine the expected price, a number of decision expressions must be defined. Some of the ones introduced before, e.g., totoalProduction are now indexed over the scenarios as well. In principle, the cost that each scenario causes is determined by the cost of constant operation, the cost of changes in the output, and the cost of violations that occur. These factors are combined to yield a final expected price. The objective is to minimize this expected price.

```
// As changing the power plants' output is expensive, the expected energy adjustment necessary to meet
// the scenarios should be as small as possible. Increasing a power plant's output is twice expensive
// as decreasing its output: price(\delta x) = 2*price(-\delta x)
dexpr float expectedPriceConstantOutput = sum (p in ControllablePlants, sp in SCENARIORANGE)
    (sum ( t in TIMERANGE ) (energyProductionScen[sp][0][p][t] +
        energyProductionScen[sp][1][p][t])/2 * ControllablePlant[p].pricePerKWh * scenarioProbability[sp]);

dexpr float expectedPriceOutputChanges = sum (p in ControllablePlants, sp in SCENARIORANGE)
    (sum ( t in TIMERANGE_SHORT )
                (abs(energyProductionScen[sp][0][p][t]-energyProductionScen[sp][0][p][t+1]) +
                  abs(energyProductionScen[sp][1][p][t]-energyProductionScen[sp][1][p][t+1]))/2
                  * ControllablePlant[p].pricePerKWh * scenarioProbability[sp]);

dexpr float expectedPriceObjective = expectedPriceConstantOutput + expectedPriceOutputChanges;

// The expected deviation of the total output from the minimum and maximum of the residual load intervals
dexpr float expectedViolationObjective = sum (sp in SCENARIORANGE)
    ((sum ( v in CLASSRANGE, t in TIMERANGE )
                abs(scenarioViolation[sp][v][t])) * scenarioProbability[sp]);

// An upper bound for the maximum costs: 2*maximum price over all scenarios and time steps + 1
dexpr float maxPrice = 1 + card(TIMERANGE) * card(SCENARIORANGE) * card(CLASSRANGE) *
    (sum( p in ControllablePlants)
        (2* ControllablePlant[p].maximalProduction * ControllablePlant[p].pricePerKWh));

// The objective is to minimize the expected violation and to minimize the price.
// However, minimizing the violation is more important.
minimize
  expectedViolationObjective*maxPrice + expectedPriceObjective/maxPrice;;
```

The constraints are similar to the ones defined before. The only difference is that they now have to take the scenarios into consideration, too.

```
// Each plants production must always be greater than its minimal production
minimalProductionConstraintSP:
  (powerPlantRunning[p][t] == true) =>
    ControllablePlant[p].minimalProduction <= energyProductionScen[sp][v][p][t];
```

Finally, some constraints have to be introduced to ensure consistency of the scenario tree and to ensure that both the lower and the upper boundary of the classes in the scenarios are adhered to.

```
// Demand must always be satisfied ( scenarioViolation [t] allows deviation from full satisfaction ).
satisfyMinDemandConstraint:
  totalProduction[sp][0][t] + scenarioViolation[sp][0][t] == residualLoad[classIdMap[sp][t]].minValue;
satisfyMaxDemandConstraint:
  totalProduction[sp][1][t] + scenarioViolation[sp][1][t] == residualLoad[classIdMap[sp][t]].maxValue;
violationUnambiguousForSpecificStateMin:
  classViolation[classIdMap[sp][t]][0] == scenarioViolation[sp][0][t];
violationUnambiguousForSpecificStateMax:
  classViolation[classIdMap[sp][t]][1] == scenarioViolation[sp][1][t];
```

Advantages and Evaluation of Trust-based Techniques

The following evaluation, taken mostly from (Anders et al., 2013a), considers techniques based on trust values alone as well as trust-based scenarios with different numbers of classes. As in the descriptions above, a single AVPP has to calculate schedules for the controllable power plants assigned to it based on

	Number of Classes	Size of Classes	μ_Δ	σ_Δ	min_Δ	max_Δ
1) Unmodified Prediction	-	-	629.1	32.8	573.6	707.3
2) Trust Value	-	-	188.9	34.9	121.2	273.2
3) Trust-Based Scenarios	$\lvert\Delta\mathcal{T}_j\rvert = 365$	≈ 50 kW	100.4	14.6	69.0	160.6
	$\lvert\Delta\mathcal{T}_j\rvert = 183$	≈ 100 kW	101.7	14.0	73.0	152.5
	$\lvert\Delta\mathcal{T}_j\rvert = 91$	≈ 201 kW	109.0	14.7	76.5	165.6
	$\lvert\Delta\mathcal{T}_j\rvert = 45$	≈ 405 kW	130.2	17.5	91.6	185.0

Table 9.2: Deviations between the actual and the expected residual load: the expected residual load is 1) equivalent to the predictor's residual load prediction ("unmodified prediction"), 2) based on a trust value, 3) based on trust-based scenarios. Data from (Anders et al., 2013a).

predictions of production from intermittent resources and predictions of consumption from consumers. Both factors are represented by a single agent, called *predictor*. Power plant models and power demand are based on real-world data from the Bavarian region Swabia[3].

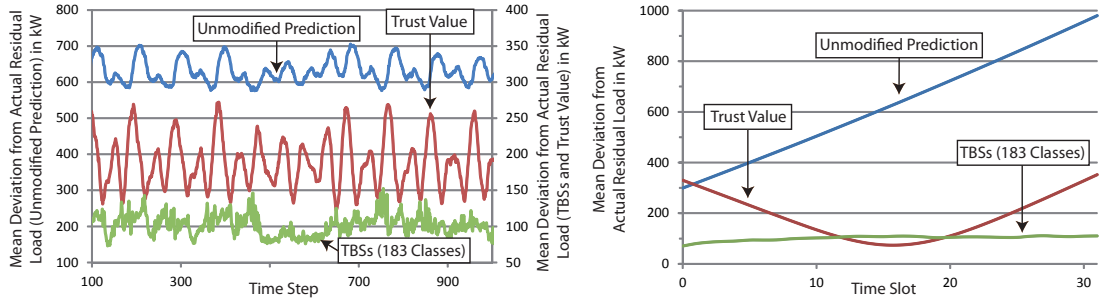
In each time step, the predictor predicted the residual load the AVPP had to satisfy in the next $n + 1 = 32$ time steps (a single time step represents 15 minutes), adding a generated prediction error to the actual residual load using a gaussian distribution The mean prediction error for a prediction in time slot 0 is 300 kW. Time-dependent behavior is implemented as the mean prediction error in time slot $j \in \{1, ..., n\}$ is 20 kW higher than the mean prediction in time slot $j - 1$. Further, the prediction error in time slot 0 depends on the most recent prediction errors for the last two time steps.

The AVPP's objective was to minimize the deviation between the actual residual load and the residual load it expected. To predict the expected residual load, the AVPP used trust-based scenarios and a trust value for comparison. The trust-based scenarios and the trust value were determined on the basis of the last $m = 50$ experiences as preliminary tests indicated that $m = 50$ provides good results for the trust value. The AVPP rated experiences and determined the trust value by means of Equation 9.2 with $\mathcal{T} = [-1, 1]$ and $k = 9125$ kW (the AVPP's maximum output/residual load). The AVPP preselects relevant trust-based scenarios on the basis of the prediction error of the last two time steps and selects a single trust-based scenarios to predict the expected behaviour by always choosing the transition with the highest probability. For each parametrization, we performed 100 simulation runs over 1000 time steps. Figure 9.13a depicts the predictor's mean behaviour over time. Figure 9.13b shows the mean prediction error for each of the 32 time slots of a prediction.

The predictions of the expected residual load are significantly improved when using trust-based scenarios instead of a trust value (see Table 9.2 and Figure 9.13a). Compared to the situation in which the AVPP relied on the residual load prediction, the mean deviation μ_Δ between the expected and the actual residual load can be reduced by approximately 70% when using trust values compared to approximately 84% when using trust-based scenarios with $\forall j \in \{0, ..., n\} : |\Delta\mathcal{T}_j| = 183$. Trust-based scenarios thus reduce the trust value's μ_Δ by 46%. In the power grid, it is of utmost importance to reduce the maximum deviation max_Δ between the expected and actual residual load. Trust values reduce max_Δ by 61%. Trust-based scenarios obtain 78% and thus max_Δ is 44% lower than the trust value's max_Δ. They also outperform the trust value in terms of minimum deviation and standard deviation. While we expected that trust-based scenarios benefit from a high number of classes $|\Delta\mathcal{T}_j|$, it is evident in Table 9.2 that 183 classes were sufficient to adequately model the underlying stochastic process. Figure 9.13b shows that trust-based scenarios could estimate the behaviour in the next 32 time steps much better than a trust value.

Summarizing, compared to trust values, trust-based scenarios significantly increase the AVPP's ability to predict the behaviour of the residual load. Moreover, the risk the AVPP is exposed to, e.g., the maximum deviations, decreases considerably and the variation in prediction quality, i.e., the standard deviation σ_Δ (see Table 9.2), declines. The latter increases the confidence in the predicted expected behaviour.

[3]Source: `http://energymap.info/energieregionen/DE/105/111.html`, visited Aug. 3, 2013; network load data from 2011: `http://www.lew-verteilnetz.de/CMS_DSO_INTER/downloads/Zeichnungen.asp`, downloaded September 9, 2012.



(a) Mean deviations from residual load.

(b) Change in deviation over time.

Figure 9.13: Mean deviation of the expected residual load (unmodified residual load prediction, trust value, trust-based scenarios) from the actual residual load, showing the accuracy of the expected behaviour for the 32 time slots and the changes in deviation over time. Images from (Anders et al., 2013a).

9.4 Solving CSPs in Open Self-Organising Multi-Agent Systems

The fact that the set partitioning problem and the hierarchical partitioning problem have both been solved with decentralised approaches (cf. SPADA and HiSPADA in Chapter 4) already indicates that it is not necessary to use a specialised constraint solver or optimiser to find solutions for the problems present in open self-organising systems if sub-optimal solutions are acceptable. On the contrary, it is seldom possible or advisable to use centralised approaches in these systems as outlined in Chapter 3. As indicated above, formalising the problem as a CSP or a CSOP can be helpful in determining the correct constraints and identifying the relevant degrees of freedom. This course of action allows a designer to "get a feeling" for the problems at hand.

The experiments detailed in Section 9.2 also show that for a larger number of agents, centralised multi-purpose algorithms can not deliver the required performance. The power plant scheduling problem has to be solved quickly—usually within 15 minutes. There is some leeway in that time horizon, especially considering that it is possible to use existing solutions as the basis for calculations and that it might not be necessary to schedule over a long period of time with the same precision as for the near future. Still, the potentially prohibitive runtime calls for other solutions that allow to solve the problem more quickly, preferably providing anytime solutions.

Centralised solutions also suffer from the need that all required information has to be available at the central point where the optimisation is performed. If this information is complex or highly volatile it might be impossible to construct a coherent overall model. The communication cost can be prohibitive and the fusion of the information and subsequent distribution of the result can add to the time required. If the information the decisions are based on changes often, the time required for information gathering and fusion can even outweigh the time for the actual solution process.

Fully decentralised solutions, however, always suffer from a lack of global information. While it is much simpler to keep localised information up-to-date, it also limits the scope of the decisions made by the agents. In essence, an agent can only decide based on the knowledge about itself and its beliefs about the states of its local environment. Usually, decisions by the individual agents in combination yield the overall result. If the decisions are not coordinated properly, they might not add up to a high-quality solution at all. This problem is exacerbated in open systems where agents make decisions mainly based on their personal interests. It is the role of *incentive-compatible mechanism design*—for a discussion of these issues in the context of power systems, see, e.g., (Fahrioglu and Alvarado, 2000)—to provide incentives in the systems that ensure that personal interest and the interest of the (sub-)system are congruent.

It is important to distinguish between a decentralised solution approach and a centralised approach that uses distributed computation. In the former, both search control and search model are distributed, i.e., no centralised model is created and no centralised algorithm distributes the work or combines partial results. MapReduce (Dean and Ghemawat, 2008) and Distributed Constraint Satisfaction (Yokoo, 2001) both distribute work and thus use distributed resources but they control the solution process centrally and often break a centralised model down into smaller parts that are then solved by the individual resources in parallel. In the latter case, decisions of the agents are only based on local knowledge

and are made without a centralised instance controlling them. While this has theoretical advantages w.r.t. scalability, it also incurs a number of issues. Termination, e.g., can often not be guaranteed—see discussion in Section 4.1.

Following Fox's (1979) suggestion of uniform hierarchies, we thus propose that *regio-central approaches* are often the most promising ones in large-scale open self-organising multi-agent systems. They combine the advantages of centralised approaches—common model and control over the solution process—with the advantages of decentralised algorithms—good scalability and localised decisions. The basic idea is that a partial solution for a specific compartment within the system is calculated by a centralised instance within that compartment. In the systems of systems context used in this thesis, such a compartment is one individual sub-system and its children. In the case of the energy management case study, the AVPP collects the information to create power plant schedules, creates the schedules centrally and distributes the schedules to its children. If the children are AVPPs themselves, they perform the same task recursively. This way, the demand that has to be satisfied is distributed from the top down, uncertainties can be handled locally, and the complex global scheduling problem is broken down into much smaller, more manageable sub-problems.

Solving problems this way, however, incurs a number of technical and conceptual challenges. The technical ones are discussed in the following two chapters. Conceptually, a system in which regio-central solutions are used can not achieve a global optimum since no global knowledge is available. Additionally, agents must be willing to divulge the information necessary to compute the solutions. While the former problem is grounded in the principles of distributed problem solving, the latter problem can be tackled, e.g., with market-based approaches such as the one discussed in (Anders et al., 2013b).

Chapter Summary and Outlook

The examples in this chapter showed how different adaptation process—ranging from self-organisation mechanisms to task scheduling—can be expressed as constraint satisfaction problems. While it is not necessarily practical to use constraint solvers at runtime, this kind of specification helps unearth important constraints and assumptions that would otherwise not show up in the requirements analysis and define a baseline standard that can, e.g., be used in evaluations. Constraint relationships allow the specification of performance criteria without the additional effort of defining pareto-optimisation models and trust values or trust-based scenarios allow to incorporate uncertainty into the models. Possibilities for decentralised solution algorithms for constraint problems are discussed as well. However, the models discussed so far are only applicable in systems that have centralised knowledge and can solve the problems centrally as well. The next two chapters therefore discuss techniques that allow the use of constraint satisfaction and optimisation algorithms in hierarchically structured systems.

Chapter 10

Accommodating Heterogeneous System Models with Synthesis of CSP Models

Summary. This chapter introduces techniques to combine several individual constraint satisfaction or optimisation models into a composed model that can then be solved by a standard constraint solver. Such a combination is necessary when solving hierarchically decomposable tasks whose solution depends on the properties of the individual agents that cooperate to solve the overall problem. Power plant scheduling in a system with a hierarchical structure is an example of such a problem and the technique is illustrated with this example. Constraint relationships allow the specification of individual optimisation criteria and an integration with an over-arching system model.

Publication. The concepts detailed in this chapter have been published in (Schiendorfer et al., 2014) and are partially based on (Schiendorfer, 2013).

Traditional divide-and-conquer approaches split a problem into its component parts and only solve those component parts. In hierarchical systems and systems of systems, this is not always possible. The problem has to be solved on different levels of abstraction on each level of the hierarchy—cf. chapter on collaborative problem solving in Wooldridge's MAS book (Wooldridge, 2008) where a hierarchical task decomposition is described that basically works like a map-reduce-approach (Luo et al., 2012). As discussed in Section 9.4, solving constraint satisfaction and optimisation problems in a regio-central fashion has advantages in large-scale open self-organising systems. If such a paradigm is employed, lower-level agents within a compartment provide information to a higher-level agent—the intermediary—that uses this information to compute a solution for the entire compartment. If the intermediary uses constraint satisfaction or constraint optimisation methods, this information can either be used as parameters for an existing model or must contain sub-models that can be combined—or synthesised—into an overall solution model.

The former variant requires a *parametrisable model* to be available. In case of the power management case study this means that a model for scheduling such as the one presented in Section 9.1 is available. The information provided by the agents then includes all fields included in powerPlantData, including the minimal and maximal output and the rate of change. This model can then be solved and the result—in this case, the schedules—can be distributed to the individual agents. Such an approach requires the agents to only divulge a limited amount of information, but also limits the expressiveness of the model as a certain homogeneity is required. For special cases—e.g., energy storages or power plants that have gaps in their possible output–the centralised model has to be refined and extended. Also, preferences of the power plants such as economic considerations or limitation to certain times of the day can not easily be incorporated.

Instead of using such a parametrised model we propose to use a *synthesised model* in which individual agent models are combined to yield a solvable regional model. As defined above, "regional" means that an intermediary collects and combines the *individual agent models* (IAM) of a certain group of other agents under its direct supervision. This allows each agent to include special constraints and individual optimisation criteria. In essence, each power plant can contribute a model that captures the operating constraints and additional issues such as economic preferences and preferences of the owner. In order to synthesise a valid *synthesised regional model* (SRM) from these IAM, they have to adhere to a

certain "interface" as defined below. Such a synthesised model can then be *abstracted* as discussed in Chapter 11 for use in a higher hierarchy layer. Therefore, the synthesis already lays the foundation for the abstraction step and relevant concepts are mentioned in this chapter accordingly.

This chapter introduces the hierarchical resource allocation problem in Section 10.1 for which the synthesis approach and the abstraction technique presented in Chapter 11 were developed. It then describes the synthesis of partial constraint satisfaction problems into a synthesised regional model in Section 10.2 before extending it with the notion of constraint relationship to allow the expression of agent-specific optimisation criteria. The techniques described in this chapter are exemplified with the decentralised power management case study. A second example of a hierarchical resource allocation problem to which they can be applied is given in the box on 147.

10.1 Hierarchical Resource Allocation Problems

The scheduling problem that presents itself in the power management system can be formalised as a resource allocation problem. The resource in this context is electric power and we refer to an allocation of the residual load to power plants for some time range as the *schedule*. This class of problems is, however, very common in multi-agent systems (Chevaleyre et al., 2006). We regard a special variant of resource allocation problems that deals with a single resource to be allocated and excludes side effects.

Definition 6 *In the* ***general one-good resource allocation problem without externalities*** *(Van Zandt, 1995), the goal is to find an allocation* $\langle x_1, \ldots, x_n \rangle$ *of the resource to* n *agents given a total quantity* x_R *of a resource, such that*

$$\underset{\langle x_1,\ldots,x_n\rangle}{\textit{minimise}} \quad \sum_{a=1}^{n} c_a(x_a) \quad \textit{subject to} \quad \sum_{a=1}^{n} x_a = x_R$$

where $c_a(x_a)$ *is a cost function for allocating* x_a *of the resource to agent* $a \in \mathcal{A}$.

Since the problem is stated to have no externalities and thus allocations do not have side effects on other agents, it can be decomposed into similar *independent* sub-problems. Therefore, agents arranged within hierarchical organizations as described in Chapter 3 can participate in the solution of the global problem in a top-down fashion. Allocating resources first to organisations of agents and then distributing the resources within the organisations recursively solves the overall problem. The hierarchical approach to the solution of these problems centralises information from a region of the system and solves this sub-problem centrally. Depending on the hierarchical structure of the system, many such regions can exist and they all solve the sub-problems concurrently. Intuitively, each AVPP in a hierarchical power plant structure can solve its regional instance of the problem in parallel with the other AVPPs on the same level of the hierarchy as depicted in Figure 10.1. These resource allocation problems and their hierarchical specialisation can be expressed with constraint satisfaction and optimization problems to make use of highly optimised general purpose constraint solvers (see, e.g., Hladik et al., 2008; Santos et al., 2002). The synthesis approach presented here assumes that the models of participating agents are available as constraint models.

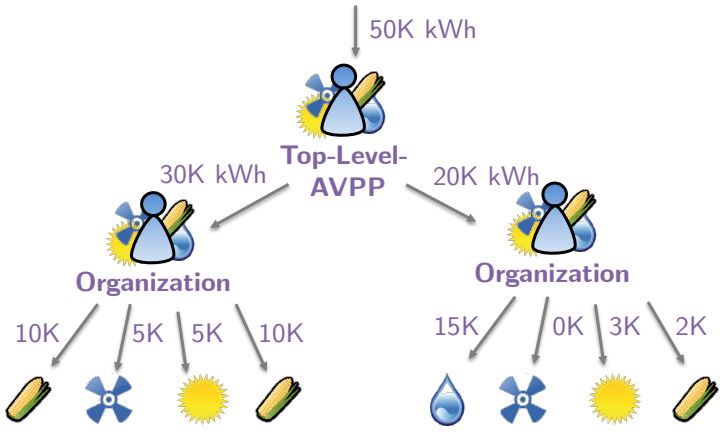


Figure 10.1: An exemplary AVPP structure solving the hierarchical resource allocation problem. The top-level AVPP has to allocate 50kW to its subordinates. These in turn have to distribute their respective share to the power plants they control. Each AVPP uses a synthesised model originating from the models provided by the controlled agents in the computation of the resource allocation.

The problems covered by our approach take into account additional aspects: In our setting, the ultimate problem consists of allocating the *production* of some resource to a set of *agents* such that a given (possibly predicted) *demand* is met by the combined production over a given *time range* as well or as cost-effectively as possible. Since meeting this demand is not always feasible, we encode the deviation of demand and production as the cost function to be optimised as opposed to stating it as a hard constraint. The possible contributions of a single agent are limited to certain ranges that can include no production at all. Additionally the allocation in earlier time steps affects future possible allocations. An allocation of resources over some time range will be referred to as *schedule*.

More formally, let $\mathcal{A}$ be a set of agents where the possible contributions to the total demand is given by L^a for each $a \in \mathcal{A}$, an ordered list of non-overlapping intervals $\in \mathbb{R}^2$ and the constraints under which the agent operates are given by $\mathcal{C}^a$. Please note that an agent's constraints have to be formulated over the only free variable $\mathcal{P}_t^a$. An agent can be excluded from the combined allocation if and only if 0 is covered by some interval. The list of intervals is equivalent to constraints limiting the possible contributions to certain regions and is motivated by the fact that practical problems may enforce minimal and maximal economical boundaries *if* any contribution is made, making the solution space for the agents non-continuous. Allocations are created for a finite time range $\mathcal{T}$ from 1 to some upper bound $max(\mathcal{T})$. Similarly, the demand that has to be satisfied is given by $\mathcal{D}_t \in \mathbb{R}$, $t \in \mathcal{T}$. The production assigned to some agent a in time step t is denoted by $\mathcal{P}_t^a$. A state $\Sigma_t^a \in \Sigma$, $t \in \mathcal{T}$, $a \in \mathcal{A}$ is a set of variables that contains all information available to a producer a up to some time step t that is required to make decisions about the production in the next time step $t+1$. In particular, Σ_t^a contains $\mathcal{P}_{t'}^a$ for all $t' \in 1, \ldots, t$. Additionally, the values of other variables such as the number of time steps an agent did not contribute can be part of Σ_t^a as they might be needed for specific constraints such as startup times. We also postulate that all constraints $c \in \mathcal{C}^a$ of agent a are limited to decision variables the agent has access to: $\forall c \in \mathcal{C}^a : sc(c) \subseteq \mathcal{X}^a$, for all $a \in Agent$.

Definition 7 *The **time-dependent non-continuous hierarchical resource allocation problem** is a variant of the general one-good resource allocation problem without externalities defined as:*

$$\begin{aligned} \underset{\mathcal{P}}{minimise} \quad & \sum_{t \in \mathcal{T}} \left| \mathcal{D}_t - \sum_{a \in \mathcal{A}} \mathcal{P}_t^a \right| \\ subject\ to \quad & \exists [x, y] \in L^a : x \leq \mathcal{P}_t^a \leq y,\ \forall a \in \mathcal{A},\ t \in \mathcal{T} \\ & \theta[sc(c)] \models c, \forall a \in \mathcal{A},\ c \in \mathcal{C}^a,\ sc(c) \subseteq \mathcal{X}^a \end{aligned}$$

with θ the assignment to all decision variables.

Such a problem can be solved in an agent hierarchy containing producers and *virtual agents* (VA)—the AVPPs in the power management case study—that do not contribute their own production but combine the production of their subordinates. An agent hierarchy $\mathcal{H}$ formally is a tree of agents and virtual agents where for some virtual agent v, $\mathsf{children}(v)$ represents the child nodes of v and $a \in \mathsf{children}(v)$ holds iff $\mathsf{parent}(a) = v$. A virtual agent and the set of all its child nodes is also referred to as "region".

A general algorithm for the time-dependent non-continuous hierarchical resource allocation problem solves it in a *top-down* fashion by first distributing the demand among the children of the root agent using a constraint solver and then have the children recursively solve their allocation problem until all leaf nodes are assigned some amount of the resource as shown in fig:stabilisation:synthesis:power-plant-structure. In light of this algorithm, the purpose and necessity of synthesis and abstraction (cf. Chapter 11) become clear. Synthesis creates an optimization problem based on models of concrete agents and abstraction leads to a simplified model used for a virtual agent on higher levels. Contrary to the demand distribution, model synthesis and abstraction are thus *bottom-up* algorithms.

Example — Load Balancing in a Server Cluster
Consider the problem of load balancing HTTP requests in a cluster of servers inspired by efforts to distribute processing capacity in grid applications investigated by Abouelela and El-Darieby (2012). Assume that "masters" are capable of assigning requests to "slaves" that handle the requests. A slave needs to communicate the minimal and maximal number of requests it may process at one time

step to its master—minimal requests are useful to justify the communication overload associated with employing a machine. The masters are organised hierarchically, where one master needs to represent the capabilities of its subordinate slaves or masters and a top level master receives the incoming requests. Upon deciding what number of requests the slaves receive to process, actual requests are distributed. Note that we do not argue that this approach is the most efficient way to solve this problem but rather shows another possible application of model synthesis and abstraction.

The set $\mathcal{A}$ consists of the slaves and masters, where the latter kind represents virtual agents that are organised in a given hierarchy. The production $\mathcal{P}_t^a$ represents the number of requests handled by agent a in time step t as a natural numbers. In this model, the states Σ_t^a track the requests currently processed to predict how many requests can be taken in step $t+1$ as requests provide their predicted duration in discrete time steps.

10.2 Specification of CSP Model Synthesis

The models involved in the synthesis process represent, on the one hand, a mathematical description of the agents' underlying physical systems and on the other hand the context for the optimisation. They can be described as follows (Schiendorfer et al., 2014):

Individual Agent Models (IAM): describe the properties of *one* concrete agent representing a physical entity in terms of constraints for the available production in $\mathcal{T}$ time steps depending on its own state (being on/off, production levels etc.) regardless of other agents. This model needs to be provided by an agent designer. An IAM defines the feasible production intervals of an agent but also regulates possible transitions between production levels at different time steps.

Organisational Optimisation Models (OOM): capture optimisation criteria and constraints imposed by the organisation that controls the virtual agent. Additional constraints usually impose certain policies of the organisation such as "prefer agents of type X" or "distribute resources evenly". Soft constraints are admissible. The OOM can also contain specialised objective functions that extend or alter the term of Definition 7 to change the type of problem to be solved.

Synthesised Regional Models (SRM): combine *several* agent models and an organisational model to yield the regio-central constraint models which describe the resource allocation problem within a *virtual agent*. The combined production is the sum of all subordinate productions. These types of models include both the actual load distribution optimization problem as well as the models used in sampling abstraction. Since preferences or soft constraints can be stated in addition to hard constraints, the resulting problems are instances of soft constraint models (Rossi et al., 2006).

The IAMs have to follow the general form of the constraints in Definition 7. In particular, the areas of feasible production have to be denoted by an interval list L^a and the constraints have to be expressed using $c_{\min}$ and $c_{\max}$. The production $\mathcal{P}_t^a$ has to be a free variable. It is possible to use constraint relationships to provide individual optimisation criteria as described in Section 10.3. A SRM for a virtual agent v results from combining a set of agent models of its children with an organisation optimisation model and is a soft constraint optimisation problem (SCOP) given by $\langle \mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{R}, f \rangle$ for some time range $\mathcal{T}$ where

- $\mathcal{X}$ are decision variables that take values from their associated domain $\mathcal{D}(x)$, for $x \in \mathcal{X}$. In particular, $\mathcal{P}_t^a \subseteq \mathcal{X}$, $t \in \mathcal{T}$, $a \in \mathsf{children}(v)$
- $\mathcal{C}$ are constraints that specify which assignments of values to variables are valid. Constraints can either be specified intensionally ($\mathcal{P}_t^a \geq 500$) or extensionally (by enumerating valid variable assignments). Hard constraints, $\mathcal{C}_h = \mathcal{C}_h^o \cup \bigcup_{a \in \mathcal{A}} \mathcal{C}_h^a$ are *mandatory* and soft constraints, $\mathcal{C}_s = \mathcal{C}_s^o \cup \bigcup_{a \in \mathcal{A}} \mathcal{C}_s^a$ should be satisfied as well as possible. $\mathcal{C}_h \cup \mathcal{C}_s = \mathcal{C}$ and $\mathcal{C}_h \cap \mathcal{C}_s = \emptyset$.
- *Constraint Relationships*, $\mathcal{R} \subset \mathcal{C}_s \times \mathcal{C}_s$ define a binary preference relation on the soft constraints as defined in Chapter 5 and discussed in Section 10.3. Each agent may provide such a relation for its soft constraints to state individual preferences. If an IAM includes soft constraints, there must be a relationship to the organisational soft constraints $\mathcal{C}_s^o$.
- f is the objective function according to which the solution is optimised. In case of the time-dependent non-continuous hierarchical resource allocation problem, the minimisation function from Definition 7 is used.

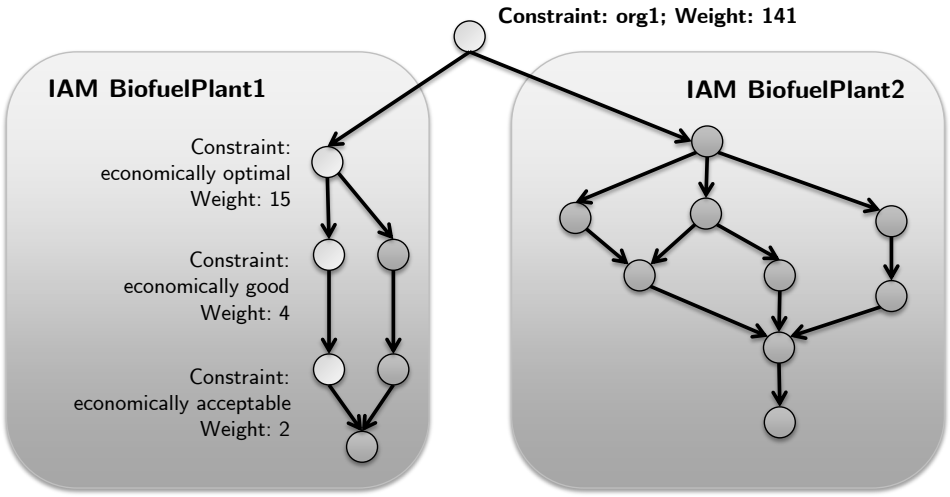


Figure 10.2: Example of the constraint relationship in a synthesised regional model with two individual agent models. Some weights for constraints in the IAM of BiofuelPlant1 for economic optimisation are shown. The individual constraints are related to an organisational constraint from the organisational optimisation model.

Information Synthesis in the Emergency Response System
The requirements of the Emergency Response System (cf. Section 2.9 and Section 14.2) call for a different kind of synthesis: information fusion. According to (Boström et al., 2007), "Information fusion is the study of efficient methods for automatically or semi-automatically transforming information from different sources and different points in time into a representation that provides effective support for human or automated decision making." Since the emergency response system calls for the combination of the personal information models of individual first responders, comprised of sensor data and the current status of the human carrying the devices that are interconnected in the system, this form of synthesis applies rather than the one described here. The main difference is that models are a very specific kind of information that is not subject to uncertainty, is clearly structured, and conforms to standards set by the system designers. Furthermore, information fusion provides the input for decision making processes, not the limitations and variables of the process itself.

10.3 Combining Individual Optimisation Criteria with Constraint Relationships

Constraint relationships are used to define soft constraints within the individual agent models and to create a relation between the constraints in the IAMs and in the organisational optimisation model. The reason for this is twofold: on the one hand, soft constraints with constraint relationships allow the agents to define optimisation criteria, e.g., for optimal production ranges or economic optimisation; on the other hand, using soft constraints avoids that the problem becomes over-constrained due to the number of constraints that become part of the problem. In general, hard constraints in the IAM should only be used to define physical limitations of the agents. In case of energy generators, this corresponds mainly to minimal and maximal output, ramp-up times, and the speed of load changes. All other constraints should be defined as soft constraints. The IAMs for power plants shown as examples in the following adhere to this rule.

Figure 10.2 shows a constraint relationship graph for a synthesised regional model consisting of two biofuel power plants. The preferences stated by the plants are incorporated into an overall graph and are related to an organisational constraint. The organisational constraint thus achieves the highest weight of all soft constraint independent of the dominance properties used (cf. Section 5.3). Since the constraints defined in the IAMs are in individual sub-trees, their weights are calculated independently, thus ensuring that both plants are treated impartially.

Optimisation of Individual Agents

Concrete soft constraints for agents participating in a hierarchical resource allocation problem with model synthesis depend greatly on the problem at hand. In a scheduling scenario in which shifts are allocated to workers, soft constraints can be used to express preferences on the workers' availability. In a classical allocation scenario where jobs are allocated to machines, preferences on maximum or minimum workload can be expressed. In the following, we will focus on power management systems and preferences for individual power plants.

Cheating
There is, of course, the chance for the owner of a power plant—or the operator of any agent participating in a time-dependent non-continuous hierarchical resource allocation problem that uses model synthesis—to cheat the system by constructing a graph with a very high path length. This ensures that the constraints defined by the operator achieve a high penalty and are therefore less likely to be violated. In essence, such a behaviour can only be effectively prevented by corresponding regulation. Technical solutions, such as capping the maximum penalty for IAM constraints, are all unsatisfactory since they might break the dominance properties or might limit the expressiveness of the IAMs.

Economic optimisation of power plant models can work in two ways: on the one hand, modulated fuel-powered generators have an economic sweet spot at which the ratio of fuel input to power output is optimal; on the other hand, spinning a generator up and down is costly since accelerating the motor costs fuel. In practice, the most effective way to run a fuel-powered generator is to accelerate it to the point of highest efficiency and leave it there. Since power plant scheduling requires degrees of freedom, however, this is not an option. But the power plants can still inform the scheduler about the parameters at which they run in an optimal way by defining respective soft constraints. For the former case, constraints that define economically optimal, economically good, and economically acceptable output ranges is a viable way to add these elements to the synthesised model. For the latter case, limiting the rate of change in a similar way is an option. Using constraint relationships to define preferences on these constraints allows to give the more important ones a higher weight and thus a higher likelihood of being fulfilled.

It is important to note that it is not possible to encode these preferences into a centralised model since they differ greatly with the type of power plant and even the type of engine and generator used. While it is not a good idea to spin fuel-powered engines up and down, running water power plants have no such limitations. It is very simple to restrict the flow of water into the generator and doing so does not imply rising costs. Therefore it is useful to encode such specialities into the IAMs where they can be adapted and optimised for the system at hand.

BiofuelPlant1 in Figure 10.2 consists of a modulating fuel-powered engine with a coupled generator that is subject to the outlined limitations of such power plants. It has an economic sweet spot that is relatively close to its maximum output—similar to car engines that produce the maximum torque and horse power close to the maximum sustainable number of revolutions. It also has a preferable rate of change that is, however, less important for the overall fuel consumption than the optimal operation range. Finally, the operator wants to limit startups and shutdowns and prefers the generator to either keep running for a longer time or stay switched off entirely.

The rate of change is limited by enforcing that changes between time steps are only within 40% of the current production.

$$\left|\mathcal{P}_t - \mathcal{P}_{t+1}\right| \leq \mathcal{P}_t * 0.4 \qquad \text{(limit rate of change)}$$

Note the use of $\mathcal{P}_t$ due to the fact that an isolated model of a single agent is regarded here. The economic considerations are captured in the following constraints:

$$900 \leq \mathcal{P}_t \leq 940 \qquad \text{(economically optimal)}$$
$$800 \leq \mathcal{P}_t \leq 950 \qquad \text{(economically good)}$$
$$700 \leq \mathcal{P}_t \leq 980 \qquad \text{(economically acceptable)}$$

Finally, two constraints are defined to prevent the power plant from starting up and shutting down repeatedly. Similarly to the economic optimisations, they are formulated as escalating constraints. The MaxSPANoperator, introduced in Section 6.2 is used here to denote that the number of consecutive shutdowns and startups should be limited. The strict version limits the number of cycles to 3 in a 10

time step window, the lenient version to 5.

$$\left[\mathcal{P}_t = 0 \wedge \mathcal{P}_{t+1} \neq 0 \vee \mathcal{P}_t \neq 0 \wedge \mathcal{P}_{t+1} = 0\right]_{10}^{3} \quad \text{(strict cycle limit)}$$

$$\left[\mathcal{P}_t = 0 \wedge \mathcal{P}_{t+1} \neq 0 \vee \mathcal{P}_t \neq 0 \wedge \mathcal{P}_{t+1} = 0\right]_{10}^{5} \quad \text{(lenient cycle limit)}$$

To yield the constraint relationship graph shown on the right-hand side of Figure 10.2, constraint relationships are formulated as follows:

$$\begin{aligned}
\text{economically acceptable} &\succ_R \text{limit rate of change} \\
\text{economically good} &\succ_R \text{economically acceptable} \\
\text{economically optimal} &\succ_R \text{economically good} \\
\text{lenient cycle limit} &\succ_R \text{limit rate of change} \\
\text{strict cycle limit} &\succ_R \text{lenient cycle limit} \\
\text{economically optimal} &\succ_R \text{strict cycle limit}
\end{aligned}$$

Relation to Organisational Constraints

The individual agents' soft constraints are required to be related to the organisations soft constraints. This way, the organisation is able to define private optimisation criteria that have precedence over the individual preferences to, e.g., ensure fairness or satisfy regulatory requirements.

In the power management case study, a number of organisational constraints are possible and different organisations can have different preferences. If an AVPP is part of a sub-hierarchy that is controlled by an organisation that emphasises ecological principles, the use of renewable resources might be preferred by stating soft constraints that assign as much of the load as possible to such power plants. If an organisation tries to maintain a large pool of reserve power, corresponding organisational constraints can be formulated. Storage power plants can also be treated preferentially this way. In the following, we use a fairness constraint that emphasises the distribution of load to all power plants. In essence, all power plants within the region controlled by the AVPP should have a similar load factor. This ensures that all power plants are working at similar capacities but does not necessarily provide economic advantages.

A simple way to state that a fair distribution of production is preferred is to calculate the load factor lf_t^a for each power plant as the quotient of the current production and the maximum production. The latter can be extracted from L^a. An organisational constraint can then be defined that states that the load factor for each power plant should be at least at a certain level:

$$lf_t^a \geq 0.4 \quad \text{(org1)}$$

The IAMs thus have to contain a constraint relationship relating their most important soft constraint to this organisational constraint. For the example above, this relationship looks as follows:

$$\text{org1} \succ_R \text{economically optimal}$$

During the synthesis process, the relationships allow constructing an overall constraint relationship graph from the constraint relationships defined in the IAMs and in the OOM. This relationship graph can then be weighted with one of the weighting functions as described in Section 5.4. The details of this process are provided in Section 10.5.

10.4 Encoding Individual Agent Models and Organisational Optimisation Models in OPL

Two version of the technical infrastructure for the implementation of the synthesis relies exist. One is integrated in the simulation environment and uses agent models provided by the power plants to create a synthesised regional model as described above. The other implementation uses OPL for the expression of individual agent models, organisational optimisation models, and the synthesised regional models. This allows the reuse of the formulation and solution techniques introduced in Chapter 9. In essence, the OPL files for the IAMs and OOMs are parsed, the relevant elements are extracted, and

the information from the models is combined to the SRM. We will use the latter implementation to describe the technical challenges and show how IAMs and OOMs can be expressed in OPL and what an SRM looks like.

Organisational Optimisation Model The OOM defines relevant constants that have to be used by the IAMs, as well as the decision variables for the overall problem. It can contain additional elements such as decision expressions and constant definitions that are not available to the IAMS. The first part of the model is the definition of the type for the list of feasible intervals.

```
/* Type definitions */
tuple IntervalType {
  float lower;
  float upper;
};
```

Constants that are part of the "external interface" include the TIMERANGE for which the system optimises. The load curve, however, is private since it only plays a role in the overall optimisation and IAMs should be formulated independent of the load. In addition, it will only be populated with predictions of the demand at runtime.

```
/* Constants available to the IAMs */
int LAST_SIMULATION_STEP = 96;
range TIMERANGE = 1..LAST_SIMULATION_STEP;

/* Private constants */
float loadCurve[TIMERANGE] = ...;
```

The model list is filled during synthesis with the power plants controlled by the AVPP. The names of the soft constraints specified in the IAMs and the OOM are also added to a corresponding set. In addition, mandatory constants are defined here that have to be provided by each IAM, including the list of feasible intervals.

```
/* Model list */
{string} plants = ...;
{string} softConstraints = ...;

/* Mandatory constants */
{IntervalType} feasibleRegions[plants] = ...;
```

Now, the decision variables are formulated. Please note that—while the abstract model overloaded $\mathcal{P}_t^a$ so that $\mathcal{P}_t^a = 0$ implied that the power plant was switched off—a specialised decision variable running is introduced here. Together with the decision expressions, this facilitates the formulation of constraints that define limitations w.r.t. ramp-up times. Penalties are included for unsatisfied soft constraints and violation is introduced as the difference between the demand and the overall production.

```
/* Decision variables */
dvar float production[plants][TIMERANGE];
dvar boolean running[plants][TIMERANGE];
dvar int+ penalties[softConstraints][TIMERANGE];

/* Decision expressions */
dexpr float max_production[p in plants] = max(f in feasibleRegions[p]) f.upper;
dexpr float loadFactor[p in plants][t in TIMERANGE] = production[p][t] / max_production[p];
dexpr float totalProduction[t in TIMERANGE] = sum (p in plants) production[p][t];

dexpr float violation = sum(t in TIMERANGE) abs(totalProduction[t] - loadCurve[t]);
dexpr float penaltySum = sum(t in TIMERANGE, c in softConstraints) penalties[c][t];
```

The OOM also contains the objective function of the AVPP. While it is possible to use arbitrary optimisations, this model minimised either the difference between demand and production or the number of violated soft constraints. The purpose of having two objective functions is discussed in Section 10.5.

```
minimize violation;
// minimize penaltySum;
```

Finally, the organisational constraints are formalised based on the decision expressions. This specific OOM does not define additional constraints apart from the organisational soft constraint. In general, arbitrary constraint can be formulated on this level.

```
/* Organisational constraints */
subject to {
  forall ( t in TIMERANGE ) {
    org1: min(p in plants) loadFactor[p][t] >= 0.4
  }
}
```

Individual Agent Models The IAMs make use of the public constants formulated in the OOMs such as TIMERANGE. Apart from that, they only specify the mandatory constants—in this case the feasible regions—and the constraints and their relationships. For the sake of simplicity, ramp-up constraints as shown in Section 9.1 have been omitted.

```
/* Feasible region */
{IntervalType} feasibleRegions = {<0, 0>, <200, 1000>};
```

The constraints use the production decision variable defined in the OOM in an altered form without indexing it over the set of plants. The synthesis algorithm changes the variable to include the power plants identifier in the system, e.g., to production["BiofuelPlant1"][t].

```
/* Generator limitations */
forall (t in 1..LAST_SIMULATION_STEP-1) {
  limit_rate_of_change: abs(production[t] - production[t+1]) <= production[t] * 0.4;
}
/* Economical considerations */
forall (t in TIMERANGE) {
  economically_optimal: 900 <= production[t] <= 940;
  economically_good: 800 <= production[t] <= 950;
  economically_acceptable: 700 <= production[t] <= 980;
}
```

Finally, the constraint relationships are defined.

```
/*
 * org1 >> economically optimal
 * economically_optimal >> economically_good
 * economically_good >> economically_acceptable
 * economically_acceptable >> limit_rate_of_change
 */
```

10.5 Execution and Evaluation of the Synthesis Process

Once the individual agent models and the organisational optimisation model are defined, they have to be combined to the synthesised region model and this model has to be solved. This section will detail the combination and the solution process and provide empirical evidence that solving a synthesised model creates solutions of a quality similar to a centralised model. The evaluation results are based on (Schiendorfer, 2013).

Synthesising OOM and IAMs

A number of provisions are made to ensure that the combination of OOMs and IAMs goes smoothly. Pre-defined constants and decision variables are used in the IAMs so that all relevant names are unique and consistent. The OOM provides a number of sets that are filled with the corresponding elements from the IAMs, including the names of the agents, the names of soft constraints, and the feasible regions.

The combination of different models requires that naming of IAM-specified constants and constraints are unambiguous. Each IAM corresponds to an agent with a unique identifier. This identifier is used

to disambiguate all names that are part of the IAMs by appending them with it. The constraint economically_acceptable shown above thus becomes BiofuelPlant1_economically_acceptable.

The overall synthesis process is structured as follows:

1. Parse the IAMs and identify constants, constraints, and the constraint relationships.
2. Rename all potentially ambiguous identifiers in the IAMs.
3. Populate the sets in the OOM that contain elements from the IAMs, including the names of agents, the names of soft constraints, and the feasible regions.
4. Construct the constraint relationship graph for the synthesised model and calculate constraint weights.
5. Alter the constraints to reflect the weights as described in Section 5.4.
6. Combine all elements and provide the synthesised regional model.

The result is a solvable synthesised regional model containing all hard constraints specified in the IAMs and in the OOM, the soft constraints including the penalties, and all elements from the OOM necessary to solve the model, including decision expressions and optimisation functions. For the example discussed so far, an excerpt of the SRM is shown below.

```
{string} plants = {"BiofuelPlant1", "BiofuelPlant2"};

{string} softConstraints = {"BiofuelPlant1_economically_acceptable", "BiofuelPlant1_economically_good",
                            "BiofuelPlant1_economically_optimal", "BiofuelPlant1_limit_rate_of_change",
                            /* [...] */ "org1"};

subject to {
  /* [...] */
  forall ( t in TIMERANGE ) {
    BiofuelPlant1_economically_acceptable:
      economically_acceptable: 700 <= production[t] <= 980
      ||
      penalties["BiofuelPlant1_economically_acceptable"][t] == 2;
  }
  /* [...] */
}
```

Optimising the Synthesised Regional Model

The optimisation of an SRM is a incremental process that requires several runs of slightly altered optimisation problems to account for the soft constraints and minimise their violation as well as the original objective. A multi-stage solution process is preferable to a pareto-optimisation since it is applicable independent of the original objective functions and there are no issues with the scaling of the individual parts of a pareto-function. Optimising an SRM $\zeta = \langle \mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{R}, f \rangle$ with respect to the original objective *and* soft constraint satisfaction can be achieved as follows:

1. Solve ζ with the original objective f and let r be the optimal result of ζ.
2. For a minimization problem, find an upper bound r' for the objective such that $r' = x \cdot r$ for some x, where $x \geq 1.0$. Note that for a non-negative minimization objective, $\lim_{r \to 0} x \cdot r = 0$ so the bound collapses to 0, implying that any valid solution has to map to the optimum if a solution with objective 0 exists. For a maximisation problem, find a lower bound and choose $0 < x \leq 1.0$ accordingly.
3. Impose a constraint to bound $f(\theta)$ with respect to r' and $\theta \in (\mathcal{X} \to \mathcal{D})$ the assignment of domain values to variables. Let ζ' be the problem that optimizes the violation of penalties with respect to the bound r': $\zeta' = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \cup \{c' : (f(\theta) \leq r')\}, \mathcal{R}, g \rangle$ where $g(\theta) = \sum_{c \in \mathcal{C}_s, \theta \not\models c} w(c_s)$ is the sum of penalties of unsatisfied soft constraints and the objective is again to minimize.
4. Solve ζ' and let p be the minimal sum of penalties of violated constraints.
5. Impose a bound on the sum of penalties to be less than or equal to p and solve for the original objective[1]: $\zeta'' = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \cup \{c' : (g(\theta) \leq p\}, \mathcal{R}, f \rangle$

[1] Note that we did not specify a tolerance for the penalty sum although this would certainly be a valid option.

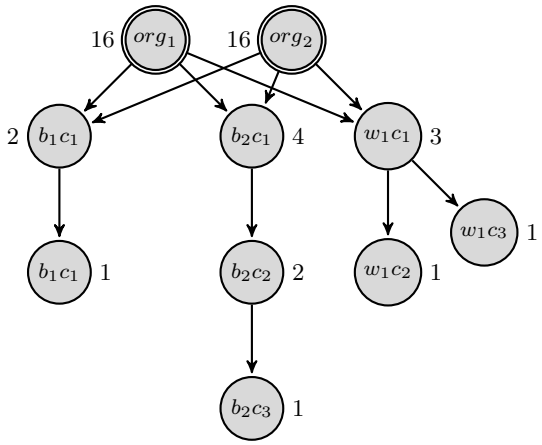


Figure 10.3: Constraint Relationship Graph of a Synthesized Regional Model consisting of an OOM with two soft constraints and three power plants with two or three constraints each.

This way, a minimal penalty for unsatisfied soft constraints is combined with a minimal objective under the circumstances. In essence, this approach constitutes a pareto-optimisation without the need of combining the different objectives in one objective function. As shown below, this approach yields solutions that optimise both organisational and individual preferences. Please note that the caveat on time-indexed constraints discussed in Section 9.2 applies to this problem as well. Therefore, the penalties decision variable is indexed over time.

Solution quality

The quality of solutions for synthesised regional models achieved with the multi-stage optimisation approach outlined above has been tested with a slightly more complicated example consisting of three power plants **b1**, **b2**, and **w1**. Each power plant either has two or three soft constraints that limit the rate of change and define the economically best production. A second organisational constraint has been introduced as well, limiting the maximum load factor and thus further promoting fairness. The constraints are shown below and the corresponding constraint relationship graph is depicted in Figure 10.3.

$b_1c_1 : \left\lvert \mathcal{P}_t^{b_1} - \mathcal{P}_{t+1}^{b_1} \right\rvert < \mathcal{P}_t^{b_1} * 0.07$	$b_1c_2 : \left\lvert \mathcal{P}_t^{b_1} - \mathcal{P}_{t+1}^{b_1} \right\rvert < \mathcal{P}_t^{b_1} * 0.1$
$b_2c_1 : 22 \leq \mathcal{P}_t^{b_2} \leq 25$ $b_2c_3 : 18 \leq \mathcal{P}_t^{b_2} \leq 33$	$b_2c_2 : 20 \leq \mathcal{P}_t^{b_2} \leq 30$
$w_1c_1 : 300 \leq \mathcal{P}_t^{w_1} \leq 350$ $w_1c_2 : \left\lvert \mathcal{P}_t^{w_1} - \mathcal{P}_{t+1}^{w_1} \right\rvert < \mathcal{P}_t^{w_1} * 0.1$	$w_1c_2 : 280 \leq \mathcal{P}_t^{w_1} \leq 370$

The time horizon of the optimisation is 11 time steps in which the power plants have to be scheduled to satisfy a demand $\mathcal{D}$. The objective function is defined as above and thus the difference between the demand and the total production has to be minimised. The IAMs are formulated in OPL and synthesised into an SRM using the process outlined. The value for x to determine the upper bound for solutions in the second stage of the optimisation was chosen as .15, allowing solutions that are at a maximum 15% worse than the optimal one found in the first stage. Table 10.1 shows the results of the first stage of the optimisation process. The optimal sum of absolute values of deviations over all time steps was found to be 293.29. The lower bound for the second optimization stage is thus $337.29 = 1.15 \times 293.29$.

The table shows that it is not possible to solve the problem without a deviation of demand and production in some time steps. It also shows that power plant b1 was not powered on in time step 0 and has to be ramped up before it can participate in the production. The rates of change of the power plants are not sufficient to deal with the peak load in time step 5, but subsequently, demand can be met. The deviations towards the end of the schedule must again be credited to the rates of change of the power plants: they can not reduce their output quickly enough so that production exceeds demand.

However, as Table 10.2 shows, the optimal solution violates several soft constraints incurring a sum of penalties for this solution of 294. In the second stage of optimisation, however, with the

upper bound for demand/production deviation set to 337.29, optimising for a minimal penalty yields a solution with a sum of penalties of only 220. The penalties for the individual constraints are shown in Table 10.4. Finally, optimizing violation with an upper penalty of bound 220 gives the solution shown in Table 10.4. The resulting solution is optimal (in terms of penalties) at 220 and reaches an aggregated demand/production deviation of 323.41 as shown in Table 10.3.

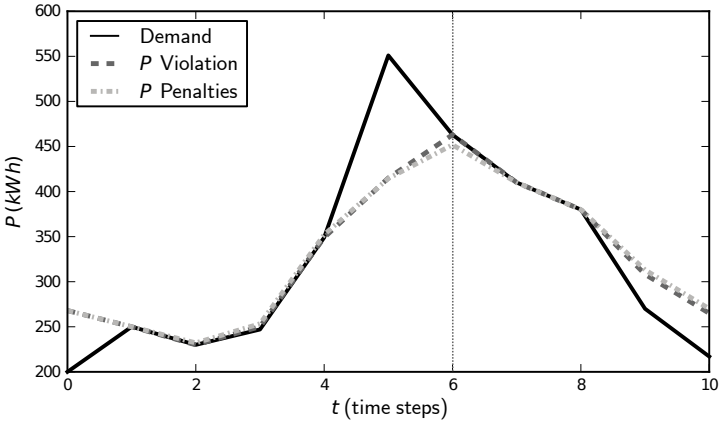


Figure 10.4: Demand and violation for the example solutions. While the production curves are practically superposable for most time steps, time step 6 shows a slight deviation introduced by the second stage optimisation for satisfaction of soft constraint satisfaction.

However, the aggregated demand/production deviation is not a good evaluation criterion for the quality. To gauge the quality of the second stage solution, the deviation in each time step is much more significant since it determines how well the AVPP fulfils its primary objective in the macro system as deviations from the demand have to be compensated in each time step. As Table 10.3 shows and Figure 10.4 illustrates, the quality of the solution optimised purely under consideration of the difference between demand and production has on average decreased by less than 1% in each time step while the penalty for unsatisfied constraints could be improved by more than 20% overall. To see how this impacts the overall solution quality, the system, and the individual agents, it is worthwhile to regard the differences of the solutions after the first stage of the process and after the second stage.

The first thing to note is that no solution is able to fulfil the organisational soft constraints in each time step. This means that the production can not be fairly distributed to the power plants. In the first four time steps, the minimal load factor constraint is violated due to the fact that power plant b1 is not running. Since that means that it is not producing any output, the constraint org1 as it has been defined above is violated. In the following time steps, the maximum load factor constraint org2 is violated since the power plants are ramping up their production at different speeds. In time step 9, the second-stage solution fulfils both organisational soft constraints by increasing b1's and b2's output while decreasing w1's output. This leads to a higher production/demand deviation which, however, stays well below 2% and is less than 5kW in total. Such minuscule differences do not impede the solution quality severely and can be easily balanced out within the overall system.

Interestingly, the other main difference between the solutions of the first and the second stage optimisation process is the fulfilment of b2's constraints. These constraints correspond to the economically acceptable, good, and optimal constraints shown before. The second-stage solution assigns a much higher production to b2 so that it can produce at the economic sweet spot most of the time after the initial ramp up and before the final ramp down. Therefore, b2 will be much more satisfied with the second-stage solution. Note that the quality of the solution for the other power plants does not change.

Chapter Summary and Outlook

This chapter discussed how models for hierarchical resource allocation problems that depend on individual agent capabilities and constraints as well as organisational specifications can be synthesised to yield solvable regio-central models. Constraint relationships can be used to express preferences of the individual agents and of the organisation. The evaluations show that the solution quality of a synthesised model is good and the technique thus provides a viable approach to the solution of hierarchical problems specified as constraint optimisation problems composed of individual agent models. The next section provides details on how synthesised models can be abstracted so that they are available as simplified versions to higher levels in the hierarchy.

t	0	1	2	3	4	5	6	7	8	9	10
$\mathcal{P}^{b1}$	0	0	0	0	50	57.5	57.5	57.5	58.82	50	50
$\mathcal{P}^{b2}$	18	17.14	15	15	16.88	18.98	18	17.14	17.14	15	15
$\mathcal{P}^{w1}$	250	232.86	215	235.1	282.13	338.55	387.5	335.36	304.03	243.23	200
$\sum \mathcal{P}_t^a$	268	250	230	250.1	349	415.03	463	410	380	308.23	265
$\mathcal{D}$	200	250	230	247	349	551	463	410	380	270	217
$\mathcal{D}_t - \sum \mathcal{P}_t^a$	-68	0	0	-3.1	0	135.97	0	0	0	-38.23	-48

Table 10.1: Optimal solution found for the objective of minimizing the difference between the demand and the total production in stage one of the solution process. The production of all three power plants is shown as well as the total production, the demand, and the deviation.

t	0	1	2	3	4	5	6	7	8	9	10	
org_1	16	16	16	16	0	0	0	0	0	0	0	
org_2	16	0	0	0	16	16	16	16	16	**16**	0	
b_1c_1	0	0	0	0	2	0	0	0	2	0	0	
b_1c_2	0	0	0	0	1	0	0	0	1	0	0	
b_2c_1	4	4	4	4	**4**	**4**	**4**	**4**	**4**	4	4	
b_2c_2	2	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	
b_2c_3	0	1	1	1	1	0	1	1	1	1	1	
w_1c_1	3	3	3	3	3	0	3	0	0	3	3	
w_1c_2	1	1	1	1	0	0	1	0	0	1	1	
w_1c_3	0	0	0	1	1	1	1	0	1	1	0	
$\sum$	42	27	27	28	30	23	28	23	27	28	11	**294**

Table 10.2: Overview of soft constraints for the first-stage solution optimising the match between demand and production. Unsatisfied soft constraints are assigned a penalty > 0. Values written in bold face discussed in the text.

t	0	1	2	3	4	5	6	7	8	9	10
$\mathcal{P}^{b1}$	0	0	0	0	50	53.5	57.25	53.24	56	52.08	50
$\mathcal{P}^{b2}$	18	20.25	20	20	22.22	25	25	23.43	24	21	20
$\mathcal{P}^{w1}$	250	229.75	212.12	233.33	280	336	369.6	333.33	300	240	200
$\sum \mathcal{P}_t^a$	268	250	232.12	253.33	352.22	414.5	451.85	410	380	313.08	270
$\mathcal{D}$	200	250	230	247	349	551	463	410	380	270	217
$\mathcal{D}_t - \sum \mathcal{P}_t^a$	-68	0	-2.12	-6.33	-3.22	136.5	11.15	0	0	-43.08	-53
Diff	0	0	2.12	3.23	3.22	0.53	11.15	0	0	4.85	5
Diff in %	0	0	0.92	1.34	0.92	0.01	2.41	0	0	1.80	2.30

Table 10.3: Optimal solution found for the objective of minimizing the difference between the demand and the total production in stage two of the solution process given an upper bound for penalties incurred by unsatisfied soft constraints. The difference to the optimal solution shown in Table 10.1 is shown as the absolute difference in the summed production per time step and in percent of the demand. The average additional deviation incurred by the second stage of the process over all 11 time steps is 0.88%.

t	0	1	2	3	4	5	6	7	8	9	10	
org_1	16	16	16	16	0	0	0	0	0	0	0	
org_2	16	0	0	0	16	16	16	16	16	**0**	0	
b_1c_1	0	0	0	0	0	0	0	0	0	0	0	
b_1c_2	0	0	0	0	0	0	0	0	0	0	0	
b_2c_1	4	4	4	4	**0**	**0**	**0**	**0**	**0**	4	4	
b_2c_2	2	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	
b_2c_3	0	0	0	0	0	0	0	0	0	0	0	
w_1c_1	3	3	3	3	3	0	3	0	0	3	3	
w_1c_2	1	1	1	1	0	0	0	0	0	1	1	
w_1c_3	0	0	0	1	1	0	0	0	1	1	0	
$\sum$	42	24	24	25	20	16	19	16	17	9	8	**220**

Table 10.4: Overview of soft constraints for the second-stage solution optimising the satisfaction of soft constraints. Unsatisfied soft constraints are assigned a penalty $\neq 0$. Values written in bold face discussed in the text.

Chapter 11

Automatic Abstraction of CSPs for Stabilisation in Hierarchical Systems

Summary. While synthesising constraint satisfaction models in a composed model allows the centralised solution of hierarchically decomposable tasks, the hierarchical nature of the systems regarded in this thesis makes it necessary to also abstract these synthesised models to allow their incorporation into the solution processes of the higher hierarchy layers. This chapter introduces the techniques used in this abstraction, ranging from simple accumulation of data to a sampling of the underlying functions to compile a more abstract, yet expressive model. We show that the combination of techniques yields an expressive model and that the abstraction error introduced is within tolerable bounds even for repeated abstractions.

Publication. The concepts detailed in this chapter have been published in (Schiendorfer et al., 2014) and are partially based on (Schiendorfer, 2013).

The synthesis approach detailed in Chapter 10 allows the combination of low-level agent models to construct solution models in higher level agents. In hierarchical systems, however, these higher-level agents are themselves children of higher level agents and thus, the synthesised models need to be further abstracted for use on a higher system level. In essence, the synthesised regional model (SRM) has to serve as an individual agent model (IAM) again. Using the synthesised model directly is not a viable option since this would incur an aggregation of complexity on the higher level. The SRMs contain decision variables defined over the constituents of the virtual agent—the AVPPs in case of the power management case study—and combining these would lead to a situation in which the decisions about the micro-level are made at increasingly higher stages in the meso-level. Therefore, the synthesised models have to be *abstracted* to allow a truly hierarchical resource allocation.

In the context considered here, abstraction is thus the process of simplifying the information contained in the model of a virtual agent to form a new model of the region represented by the VA which is then provided to the VA's father. In this way, higher levels are not overwhelmed with the details of the lower levels but can still provide solutions that can actually be achieved by the lower levels. Our understanding of abstraction follows (Giunchiglia and Walsh, 1992): "Abstraction is the mapping of a problem representation into a simpler one that satisfies some desirable properties in order to reduce the complexity of reasoning."

In addition to the individual agent models (IAMs) and the synthesised regional models (SRM) we introduce abstracted agent models (AAMs). AAMs are simplified agent models of a set of underlying agents that capture their essential properties and serve as models of virtual agents that can be provided to a superior instance. Since IAMs and AAMs have the same structure, it does not matter to this higher instance whether it is working with a micro-level agent and thus an individual agent or a virtual meso-level agent and thus an already abstracted model. Much more than the synthesis process, the abstraction process and the accompanying simplifications will lead to errors that can either be mitigated with a reactive negative feedback mechanism (cf. Section 8.4) or by externalising the error, e.g., on a power market in the case study or by passing it on to the next level.

Since the AAM should be equivalent to an IAM with regard to the way it is handled by a superordinate agent, an AAM likewise defines production ranges and the rate of change. In essence, it is indistinguishable from an IAM by providing the same "interface". Of course, this information needs to

be preserved in the abstraction process and distilled from the synthesised model in a robust fashion. In addition, the AAM contains constraints that express the behaviour of the collective that is captured in the synthesised model and is thus controlled by the VA. The abstraction process is concerned with finding the "corners" of the possible behaviours as well as "holes" in the possible contribution, i.e., the extrema of what the VA can contribute by scheduling the collective appropriately and those areas that can not be reached by a schedule. In addition, time-dependent and non-linear processes such as rates of change or cost functions have to be captured and approximated with appropriate constraints.

In the following, we will describe the abstraction process that creates AAMs from SRMs. The first step is to combine the production intervals of the feasibleRegions as described in Section 11.1. Then, temporal dependencies in the SRMs will have to be abstracted as explained in Section 11.2. The third step is the abstraction of potentially non-linear functional relationships with sampling as described in Section 11.3. The approach is evaluated for its effects on solution times and quality in Section 11.5. Related work is summarised in Section 11.6. The running example used in the following is taken from (Schiendorfer et al., 2014).

11.1 General Abstraction to Combine Sets of Production Intervals

The production a VA can offer depends on the production offered by the individual agents it controls. General boundaries on this production, independent of the rate of change, need to be combined to abstract the diversity of the underlying agents. Possible production ranges are provided by the agent models as intervals. For a physical, controllable power plant, the possible contribution can most times be expressed by two intervals, e.g., $\{[0,0],[1,4]\}$, where the first item describes that the plant can be shut down and the second that if the plant is running, it has to produce between 1kw and 4kw. For other VAs, these intervals can be more complex due to the general abstraction process as described below.

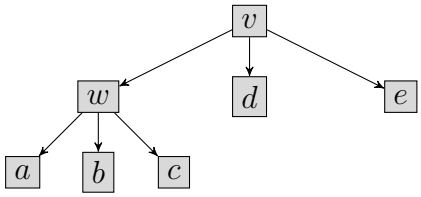
Given a VA that controls the physical power plant mentioned above, as well as a second one with an interval of $\{[0,0],[7,10]\}$, different actual productions can be satisfied by the VA. If one of the power plants is shut down, the respective intervals from the individual plants can be produced. If both power plants are running, any production in the range $[8,14]$, calculated by summing minimal and maximal production, can be achieved. Thus, the possible productions of the VA that must be reflected in the AAM are $\{[0,0],[1,4],[7,10],[8,14]\}$. Of course, these intervals can be further simplified. It is important to note, however, that there is a production gap in the interval $(4,7)$ that can not be covered by this particular VA and that thus should not be assigned by the superordinate agent.

Since the VA acts as a façade for the sub-tree it spans, it abstracts from the way an allocated production will be distributed to the individual agents. For the agent controlling the VA it does not matter, whether, e.g., an output of 8 will be allocated internally by setting $\mathcal{P}^1$ to 1 and $\mathcal{P}^2$ to 7 or by $\mathcal{P}^1$ to 0 and $\mathcal{P}^2$ to 8. Therefore, overlapping intervals can be contracted, so that $[7,10]$ and $[8,14]$ become $[7,14]$. The feasible regions of the exemplified VA are thus provided as $\{[0,0],[1,4],[7,14]\}$; there is a unique production "hole" that can not be internally allocated at $(4,7)$ and 0 and 14 constitute minimal and maximal production.

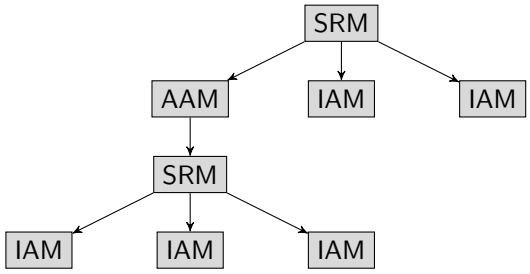
Our goal to describe the possible productions can thus be achieved by using *general abstraction* as demonstrated by the example above. A simple formulation of the process makes use of an operator $\oplus$ that signifies the plus-operation in interval arithmetic such that $[x_1,y_1]\oplus[x_2,y_2]=[x_1+x_2,y_1+y_2]$ to calculate the possible combined production of two agents. The recursively defined contraction-operation $\Downarrow$ takes a sorted list of intervals (in ascending order of the lower bounds of the intervals) and merges overlapping intervals such that, e.g., $\langle[1,5],[4,7]\rangle \Downarrow = \langle[1,7]\rangle$. Concatenation of lists is denoted by $L_1 + L_2$.

$$\langle\rangle \Downarrow = \langle\rangle$$
$$\langle[x,y]\rangle \Downarrow = \langle[x,y]\rangle$$
$$(\langle[x_1,y_1],[x_2,y_2]\rangle + L) \Downarrow = \begin{cases} \langle[x_1,y_1]\rangle + (\langle[x_2,y_2]+L)\Downarrow) & \text{if } y_1 < x_2 \\ (\langle[x_1,\max\{y_1,y_2\}]\rangle + L)\Downarrow & \text{else} \end{cases}$$

Each agent a that is part of a Virtual Agent provides a sorted list L^a of non-overlapping intervals that indicate its feasible regions. The combined list of feasible regions for a VA is computed by combining all intervals of the agents, sorting the result by the intervals' lower boundaries, and contracting the sorted list:



(a) Hierarchy of power plants and AVPPs.



(b) Models involved in the synthesis and abstraction.

Figure 11.1: Simple structure of agents and virtual agents used to demonstrate the abstraction process. Agents a, b, c, d, and e are micro-level agents that provide IAMs, v and w are virtual agents where w provides an AAM to v. The synthesis example in Section 10.5 created an SRM for w. The corresponding hierarchy of models is shown on the right hand side.

$$L^a \oplus L^b := \mathsf{sort}(\{L_i^a \oplus L_j^b \mid 1 \leq i \leq |L^a|, 1 \leq j \leq |L^b|\}) \Downarrow$$

For application to finite sets of lists $\mathcal{L}$, the $\oplus$ operation is extended as $\bigoplus_{L^i \in \mathcal{L}} L^i = L^1 \oplus (L^2 \oplus (\dots L^n))$. For a VA v, we write $L^v = \bigoplus_{a \in \mathsf{children}(v)} L^a$. The results of this general abstraction can then be used as a constraints to limit the feasible regions of a VA:

$$\forall t \in \mathcal{T} : \exists [x, y] \in L^v : x \leq \mathcal{P}_t^v \leq y$$

A simple hierarchy of power plants and AVPPs (which constitute VAs) is shown in Figure 11.1a. We will illustrate how general abstraction can be applied to create a list of feasible intervals for the VA w that contains only power plants and does not control other VAs. Each of the power plants a, b, c can produce within a continuous range and can be switched of. Therefore, 0 is a valid overall contribution of w. If we assume certain production ranges for the three power plants, $L^a = \langle [0,0], [50,100] \rangle$, $L^b = \langle [0,0], [15,35] \rangle$, and $L^c = \langle [0,0], [200,400] \rangle$ are valid intervals. For w, the combination of these three intervals by the abstraction process yields:

$$\begin{aligned}
L^w &= L^a \oplus L^b \oplus L^c \\
&= L^a \oplus (\langle [0,0], [15,35] \rangle \oplus \langle [0,0], [200,400] \rangle) \\
&= L^a \oplus (\mathsf{sort}(\{[0,0] \oplus [0,0], [0,0] \oplus [200,400], [15,35] \oplus [0,0], [15,35] \oplus [200,400]\}) \Downarrow) \\
&= L^a \oplus (\mathsf{sort}(\{[0,0], [200,400], [15,35], [215,435]\}) \Downarrow) \\
&= L^a \oplus (\langle [0,0], [15,35], [200,400], [215,435] \rangle) \Downarrow) \\
&= \langle [0,0], [50,100] \rangle \oplus \langle [0,0], [15,35], [200,435] \rangle \\
&= (\langle [0,0], [15,35], [50,100], [65,135], [200,435], [250,535] \rangle) \Downarrow \\
&= \langle [0,0], [15,35], [50,135], [200,535] \rangle
\end{aligned}$$

This result is a valid abstraction of the capabilities of the children of w and can be communicated to the superordinate AVPP v and used there in creating a schedule for w. v can combine L^w—a list of intervals—with the intervals communicated by the power plants d and e. Thus, v can be completely oblivious to the type of agent controlled by it. It can treat both VAs and micro-level agents the same way when it synthesises the model used in its internal scheduling process and in creating an abstracted model of itself. Assuming that plant e *must* produce between 14 and 20 and can not be disconnected while d offers the intervals $[0,0]$ and $[6,10]$, the abstraction of v yields:

$$\begin{aligned}
L^v &= L^w \oplus L^d \oplus L^e \\
&= L^w \oplus (\langle [0,0], [6,10] \rangle \oplus \langle [14,20] \rangle) \\
&= L^w \oplus (\langle [14,20], [20,30] \rangle) \Downarrow \\
&= \langle [0,0], [15,35], [50,135], [200,535] \rangle \oplus \langle [14,30] \rangle \\
&= (\langle [14,30], [29,65], [64,165], [214,565] \rangle) \Downarrow \\
&= \langle [14,165], [214,565] \rangle
\end{aligned}$$

The list of feasible intervals L^v is thus a valid representation of the capabilities of v that matches the limitations set by the individual power plants. The least possible contribution must be 14 since e can not be switched off and must provide at least 14 and the maximal contribution is the sum of all maximal contributions of the power plants a to e. Power plants d and e also contribute power that allows "closing" some of the supply holes that were not covered by the children of w, thus leading to a more compact representation of the production space.

11.2 Abstraction of Processes with Temporal Dependencies

The general abstraction of the feasible regions of a VA only supplies constraints for possible production of the VA. It does not take into account that the actual production in each time step can depend on the current state of the system if, e.g., rate of change is restricted or startup times have to be observed. To cover such cases, temporal dependencies have to be considered to determine the feasible regions after t time steps *from now on* given some initial state Σ_0.

For a VA v, the production level $\mathcal{P}_t^v$ after t time steps only depends on the feasible productions of its children after t steps. Thus, it is necessary to determine the minimal and maximal state $\Sigma_{\mathrm{m},t}^a$ with $\mathrm{m} \in \{\min, \max\}$ for all agents $a \in \mathsf{children}(v)$ in time step t starting from the current Σ_0. These boundaries are determined by creating "virtual" schedules that minimise (or maximise) the output of each power plant starting from the current state and respecting all constraints. The outputs of these virtual schedules can then be combined as described above for each time step to determine the corridor in which the schedules can be created. The results of such a process are illustrated in Figure 11.2.

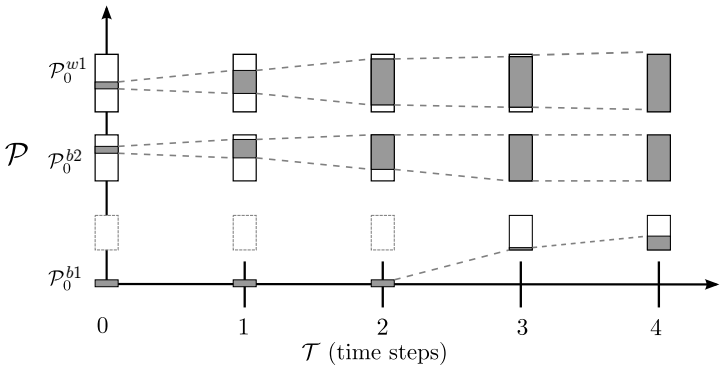


Figure 11.2: Temporal abstraction for a virtual agent consisting of three agents. White boxes indicate general bounds, grey areas represent the boundaries at step t. Agent 1 needs two time steps to start up and is then available at its minimum output. Note that only the feasible regions according to the selection of a production in $t = 0$ are regarded. Picture from (Schiendorfer et al., 2014).

Temporal abstraction is an efficient way to reduce the search space by prohibiting solutions that are not achievable given, e.g., a rate of change or other time-dependent constraints. In theory, sampling abstraction (cf. Section 11.3) can achieve similar results but is not as efficient. However, for temporal abstraction to work, relevant constraints have to be formulated accordingly. The rate of change constraint of a physical power plant, e.g., was formulated in Section 10.4 as follows:

```
/* Generator limitations */
forall (t in 1..LAST_SIMULATION_STEP-1) {
  limit_rate_of_change: abs(production[t] - production[t+1]) <= production[t] * 0.4;
}
```

To make this constraint available to the temporal abstraction process outlined below, it instead has to be expressed as a pair of functions, $c_{\min}, c_{\max} : \Sigma \to \mathbb{R}$, that restrict the possible allocations in a following time step based on the current allocation. A maximal relative change $|\mathcal{P}_t^a - \mathcal{P}_{t+1}^a| \leq \mathcal{P}_t^a \times x$ could for instance lead to the functions $c_{\min}(\mathcal{P}_t^a), c_{\max}(\mathcal{P}_t^a) = \mathcal{P}_t^a \pm \mathcal{P}_t^a \times x$. In OPL:

```
/* Generator limitations */
forall (t in 1..LAST_SIMULATION_STEP-1) {
  limit_rate_of_change_min: production[t+1] <= production[t] - production[t] * 0.4;
  limit_rate_of_change_max: production[t+1] <= production[t] + production[t] * 0.4;
}
```

If the constraints of a concrete problem instance are available in this form or can easily be mapped to such functions, Algorithm 2 can be used to exclude infeasible parts of the search space efficiently. For a

future time step t, we identify the minimal and maximal contribution of each $a \in \mathsf{children}(v)$ with respect to all its constraints $\mathcal{C}^a$ by taking the minimal maximum value as the upper bound and analogously the maximal minimum value as the lower bound. To get the minimum and maximum contributions of the VA in time step t, we combine and merge the resulting intervals (line 8). Furthermore, the minimal and maximal bounds are stored in $\Sigma^a_{\min}$ and $\Sigma^a_{\max}$ to calculate similar limitations for time step $t+1$ (lines 9 and 10). L^v_t represents the feasible regions of v after t steps and thus corresponds to the merged grey intervals in Figure 11.2.

Algorithm 2 Temporal Abstraction to exclude infeasible ranges

1: **procedure** TEMPORAL-ABSTRACTION(v, Σ_0)
2: $\quad\forall a \in \mathsf{children}(v) : \Sigma^a_{\min}, \Sigma^a_{\max} \leftarrow \Sigma^a_0$
3: $\quad$**for all** $\{t \in \mathcal{T}\}$ **do**
4: $\quad\quad\mathcal{I} \leftarrow \emptyset$
5: $\quad\quad$**for all** $\{a \in \mathsf{children}(v)\}$ **do**
6: $\quad\quad\quad\mathcal{P}^a_{min} \leftarrow \mathsf{max}(\{c_{\min}(\Sigma^a_{\min}) : c \in \mathcal{C}_a\})$
7: $\quad\quad\quad\mathcal{P}^a_{max} \leftarrow \mathsf{min}(\{c_{\max}(\Sigma^a_{\max}) : c \in \mathcal{C}_a\})$
8: $\quad\quad\quad\mathcal{I} \leftarrow \mathcal{I} \uplus \{[\mathcal{P}^a_{min}, \mathcal{P}^a_{max}]\}$
9: $\quad\quad\quad\Sigma^a_{\min} \leftarrow \Sigma^a_{\min} \cup \{\langle t, \mathcal{P}^a_{min}\rangle\}$
10: $\quad\quad\quad\Sigma^a_{\max} \leftarrow \Sigma^a_{\max} \cup \{\langle t, \mathcal{P}^a_{max}\rangle\}$
11: $\quad\quad$**end for**
12: $\quad\quad L^v_t \leftarrow \bigoplus_{i \in \mathcal{I}} i$
13: $\quad$**end for**
14: $\quad$**return** $\{L^v_t \mid t \in \mathcal{T}\}$
15: **end procedure**

To illustrate the concept of such constraint functions, consider the following constraints for a fixed rate of change: $c_{\min}(\Sigma^a_t) := \max(0, \mathcal{P}^a_t - \mathcal{P}^a_\delta)$ and $c_{\max}(\Sigma^a_t) := \min(\mathcal{P}^a_t + \mathcal{P}^a_\delta, \mathcal{P}^a_{max})$. Of course, if the behaviour of the power plant is more complex (e.g., a minimal production greater than 0 and the possibility to switch the power plant off exists), these function could be more complex. From these constraints and their use in the creation of the "virtual" schedules that minimise and maximise production in those time steps, feasible regions of $\mathcal{P}^v_t$ are calculated that are easily integrated with the AAM found by general abstraction and further reduce the abstracted search space.

$$\forall t \in \mathcal{T} : \exists [x, y] \in L^v_t : x \leq \mathcal{P}^v_t \leq y$$

The longer the time horizon (i.e., for higher values of t, the higher the probability that temporal boundaries coincide with the general boundaries. It is therefore possible to only create "virtual" schedules until these general boundaries are reached. In the example in Figure 11.2, the general boundaries for agents 1 coincide with the boundaries found by temporal abstraction in time step 4 and for agent 2 in time step 3.

Applying Temporal Abstraction to the Power Management Case Study

Based on the power plant data used for the evaluation of model synthesis in Section 10.5, a few steps of the temporal abstraction algorithm are outlined below to illustrate the concept for the virtual agent w. The illustration has been adapted from (Schiendorfer, 2013). At $t = 0$, the initial state, $\mathcal{P}^a_0 = 0$, $\mathcal{P}^b_0 = 17$, and $\mathcal{P}^c_0 = 300$. Thus, both plants 2 and 3 are running while plant 1 is currently switched off. To find the boundaries for the following time steps, we apply Algorithm 2 for minimisation and maximisation. The example also considers switching power plants on and off. Therefore, a couple of additional constraints need to be defined. First of all, directly after a power plant has been switched on, it produces its minimal output:

```
minimal_production_after_switch_on:
  (running[t] == false && running[t+1] == true) => production[t+1] == 50;
```

Secondly, the number of time steps the power plant has been in the "off" state must exceed the minimum number of time steps minOffTime the power plant has been in the off state. This constraint enforces that after shutting a thermal power plant down, the plant has to stay off for a while to avoid constant restarts. It uses the decision expression consecutivelyStopped which determines the number of time steps the power plant has been off.

```
minimal_standing_time:
  (running[t] == false && running[t+1] == true) => (consecutivelyStopped[t] − minOffTime) >= 0;
```

Note that in the following, σ_t^a holds information about whether agent a can potentially be on or off in time step t. The number of time steps an agent a has been switched on consecutively is stored in ν_t^a, the number of time steps it has consecutively been switched of in χ_t^a. Initially, σ_0^a corresponds to $\{\nu_0^a\}$ for agent a. The minimal off time for all agents in the example is two time steps. Agent $b1$'s off time at time step 0 $\chi_0^b 1$ is 1. Rates of change are given as percentage measures of the current output in $\mathcal{P}_\delta^a$ for an agent a.

First Time Step In the first time step $t = 1$, the minimal_standing_time constraint prevents plant $b1$ from being switched on as it has only been off for one time step in $t = 0$. Instead, $|\chi|_1^{b1}$ is set to 2 and stored in Σ_1^{b1} which serves as a "countdown". Plant $b2$ initially runs at a production of 17. With a relative $\mathcal{P}_\delta^{b2}$ of 15%, it may vary its production by 2.55, hence reaching values from 14.45 to 19.55. The general boundary constrains this output to be greater than or equal to 15 which is why it is necessary to take the maximum of the minimally reached values as defined in line 6 of Algorithm 2. For plant $w1$, according to the rate of change of 20%, all contributions between 240 and 360 are possible. Plant $b2$ reaches its minimum output at this step so in the *next* step a transition to off is possible. Similarly plant $b1$ needs to wait one more step to begin contributing, thus $\sigma_1^{b1} = \{0\}$, $\sigma_1^{b2} = \sigma_1^{w1} = \{1\}$. In summary:

$$\begin{aligned} L_1^w &= L_1^{b1} \oplus L_1^{b2} \oplus L_1^{w1} \\ &= \langle [0,0] \rangle \oplus (\langle [15, 19.55] \rangle \oplus \langle [240, 360] \rangle) \\ &= \langle [255, 379.55] \rangle \end{aligned}$$

Second Time Step In $t = 2$, plant $b1$ has been disconnected for 2 steps, which is the minimal number of standing steps required by minimal_standing_time. Therefore it may contribute at its minimum production in $t = 2$ for maximisation but of course can also be left switched off for the minimisation step. Plant $b2$ can be disconnected completely or contribute in any of $[15, 22.48]$ and plant $w1$ arrives at its upper limit 400 (432 by relative change) and at 200 (192) for minimization—hence, it converges to its general boundaries at $t = 2$ but still has to run in this step. Now we have $\sigma_2^{b1} = \sigma_2^{b2} = \{0, 1\}$, $\sigma_2^{w1} = \{1\}$ and consequently:

$$\begin{aligned} L_2^w &= L_2^{b1} \oplus L_2^{b2} \oplus L_2^{w1} \\ &= \langle [0,0], [50,50] \rangle \oplus (\langle [0,0], [15, 22.48] \rangle \oplus \langle [200, 400] \rangle) \\ &= \langle [0,0], [50,50] \rangle \oplus \langle [200, 422.48] \rangle \\ &= \langle [200, 472.48] \rangle \end{aligned}$$

Third Time Step In the third time step $t = 3$, plant $b1$ can be allocated a higher production since it has potentially been started already. Possible intervals are now $[0, 0]$—in case it has indeed not been switched on—and $[50, 56.25]$. Plant $b2$ can be utilised at a capacity of up to 25.29. Since plant $w1$ has reached its minimum and maximum production levels in the previous step, it may now be switched of or contribute within *any* of its general range boundaries. Now, $\sigma_3^{b1} = \sigma_3^{b2} = \sigma_3^{w1} = \{0, 1\}$, and by combining them:

$$\begin{aligned} L_3^w &= L_3^{b1} \oplus L_3^{b2} \oplus L_3^{w1} \\ &= \langle [0,0], [50, 56.25] \rangle \oplus (\langle [0,0], [15, 25.29] \rangle \oplus \langle [0,0], [200, 400] \rangle) \\ &= \langle [0,0], [50, 56.25] \rangle \oplus \langle [0,0], [15, 25.29], [200, 425.29] \rangle \\ &= \langle [0,0], [15, 25.29], [50, 56.25], [65, 81.54], [200, 481.54] \rangle \end{aligned}$$

The process continues for further steps until all plants have reached the general boundaries. Then L_t^w converges to L^w and the constraints already established by general abstraction suffice to exclude infeasible production ranges.

11.3 Abstraction of Non-Linear Processes with Sampling Point Approximation

Temporal abstraction relies on the the ability to formulate constraints as $c_{\min}$ and $c_{\max}$ as illustrated above. However, constraint optimisation processes can contain constraints that can not be formulated this way. In addition, it is desirable to approximate potentially non-linear relationships within a synthesised regional model such as production costs that depend on the the sum of the individual production costs and the way the production is distributed to the individual agents. If economical consideration have to be captured on a higher level, these costs have to be made available in AAMs. The rate of change is also more complicated than temporal abstraction might suggest: while temporal abstraction captures the corridor in which the production of agents can develop in the next time steps, it does not assert the actually achievable output in t+2 for a specific production in t+1.

Therefore, an additional technique has to be employed that *samples* the underlying SRM for specific productions and attains the relevant results for the function of interest. More concretely, variants of an SRM are created that define constant productions and contain specific objective functions. The results of the solution process, especially the result of the objective function, are then used in the definition of piecewise linear functions (PWL) that approximate the behaviour of the non-linear process in the SRM. PWLs are readily supported by mixed-integer programming algorithms or constraint solvers such as CPLEX and have been applied in model abstraction in simulation engineering (Frantz, 1995). Algorithm 3 shows the general procedure of sampling abstraction. If the rate of change has to be sampled, the procedure can be called with maximize $\mathcal{P}^v_{t+1}$ and minimze $\mathcal{P}^v_{t+1}$ to explore the possibilities of both minimising productions in future time steps and maximising it. In case of maximisation, $\mathcal{IO}$ corresponds to $\mathcal{P}^{v+}_{\delta}$ and the resulting PWL contains a function mapping production in each t to the maximally attainable production in $t+1$.

Algorithm 3 Sampling Abstraction

Require: $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ is the SRM of v
Ensure: $\mathcal{IO}$ are pairs of input/output values

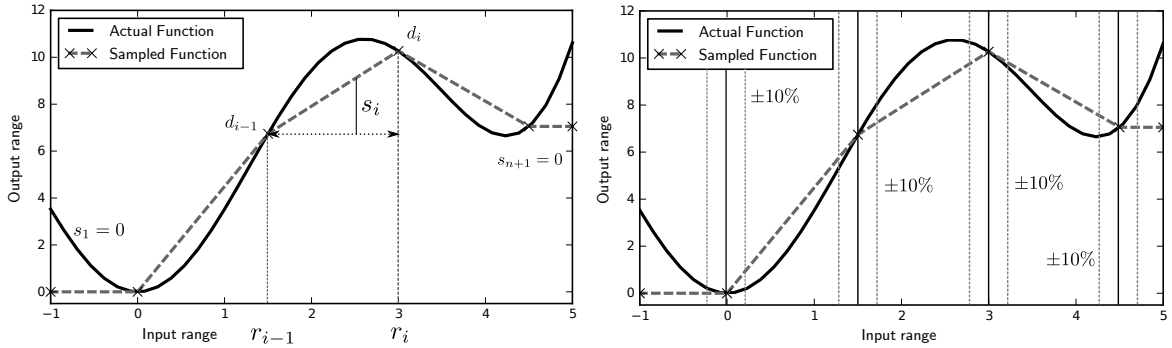
```
1:  procedure SAMPLING-ABSTRACTION(v, f)
2:      for all {t ∈ T} do
3:          S ← sampling points ∈ L_t^v
4:          for all {s ∈ S} do
5:              C' ← C ∪ {(P_t^v = s)}
6:              o ← solve ⟨X, D, C', f⟩
7:              IO ← IO ∪ {(s, o)}
8:          end for
9:      end for
10:     return pwLinear(IO)
11: end procedure
```

In essence, the sampling approach thus tests different possibilities for future development based on potential decisions for the present. A vital variable in the process is the selection of the tested potential decisions and thus of the sampling points. Currently these sampling points are selected equidistantly across the full range of values provided by the intervals in L^v_t. The approach is sketched for both minimisation and maximisation of the power plant production consistent with the rate of change in Figure 11.3.

The piecewise linear function that results from the sampling abstraction provides an approximation of the underlying process behaviour. Figure 11.4a illustrates the approximation of a non-linear function with a—rather crude—piecewise linear function. To make the PWL that sampling creates usable in a solver such as CPLEX, some conventions were defined. The function's value below the minimal and above the maximal input values are equal to those sampled extreme values, thus: $s_1 = s_{n+1} = 0$. The set of break points T corresponds to the sampled inputs. Slopes are found by interpolating between the collected corresponding output values. Let $r_i \in R$ be the n input values (range) and $d_i \in D$ be the n output values (domain) such that $F(r_i) = d_i$. Then, for all $i \in [2, n]$ we have that $s_i = \frac{d_i - d_{i-1}}{r_i - r_{i-1}}$. Again, by convention, we designate (t_0, v_0) as (r_1, d_1) but could use any of the sampled input-output pairs.

To allow exploration of the solution space, the generated problems are made more flexible. Concretely, after finding the designated sampling points in the range of the function (i.e., the production

Figure 11.3: Sampling of the possible future production changes for power plant $b2$ for three sampling points based on the data from Figure 11.2. Three sampling points are selected to check for the possible productions depending on the choice of production in time step $t+1$. While the entire corridor calculated by temporal abstraction is covered, different choices in $t+1$ restrict the possibilities in $t+2$. If the middle sampling point is chosen, e.g., only the chequered area contains valid productions for the next time step.



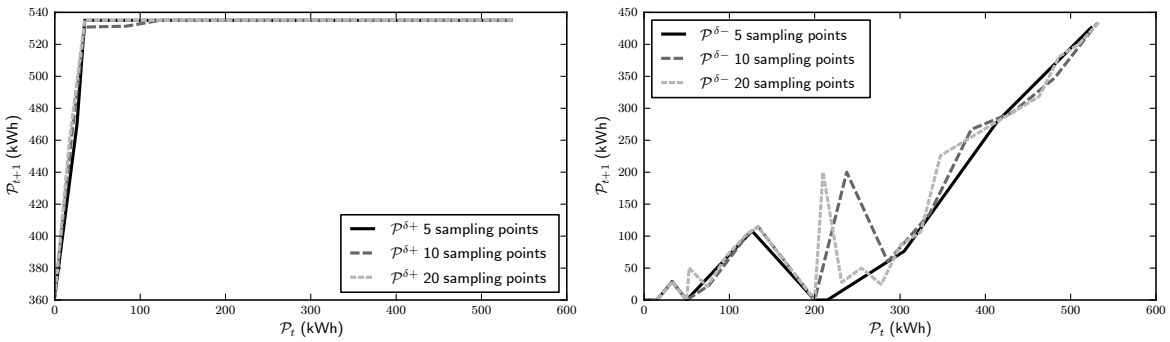
(a) Strict input values.

(b) Flexible input values.

Figure 11.4: Example function $f(x) = x^2 + 3x\sin(x)$ sampled at $\{0, 1.5, 3.0, 4.5\}$, both with strict input values and flexible ones. The flexible input values allow exploration of the sampling space. Figures adapted from (Schiendorfer, 2013).

space), for each optimisation problem the input values can be in a corridor of 10% around the actual sampling point, as depicted in Figure 11.4b, by not fixing the sampling with, e.g., $p_t = 500$ but by using $490 \leq p_t \leq 510$. In some circumstances, this allows finding solutions much more quickly, e.g., because the solution space becomes larger and more solutions exist. However, caution has to be taken when the resulting function is extensive (i.e., $x \leq f(x)$). Randomising the input as sketched makes solutions with an input lower than the output valid in some cases. For instance, when maximizing $\mathcal{P}_1^v$, a solution with input $\mathcal{P}_0^v < \mathcal{P}_1^v$ can be a valid solution and even the optimal solution. From this, peculiarities such as a maximal production of 75000 in the next step when the current production is 80000 could arise. Inserting a constraint $\mathcal{P}_1^v \geq \mathcal{P}_0^v$ explicitly excludes such a condition and asserts the extensivity. Similarly, results from previous outputs of sampling points can be used to ensure monotonicity.

We sketch the approach in Algorithm 3 for the production change speeds $\mathcal{P}_\delta^{v-}$ and $\mathcal{P}_\delta^{v+}$. We are interested in the maximal and the minimal production $\mathcal{P}_{t+1}^w$ given $\mathcal{P}_t^w$. In the energy example, the positive change function exhibits two properties of interest that are captured when creating sampling models: extensivity ($f(x) >= x$, as a plant may always stay at the same production level) and monotonicity ($x \leq y \rightarrow f(x) \leq f(y)$ which is assumed for the AVPP case). The resulting PWL is shown in Figure 11.5a. In contrast to the positive change, those properties do not necessarily hold for the negative case in the AVPP example. Consider a simple AVPP consisting of two plants, one of which having the available regions $[0, 0], [50, 100]$, $\mathcal{P}_\delta = 5$ and the other one providing feasible regions of $[0, 0], [200, 400]$. Then the least production level reachable at the current level of 70 is 65 whereas the AVPP can be disconnected if it is running at 200. This circumstance is also present in Figure 11.5b. The "sampled" piecewise linear functions can then be used for additional constraints in the AAM:



(a) Positive Rate of Change.

(b) Negative Rate of Change.

Figure 11.5: Positive and negative rate of change as a piecewise linear function of the current production obtained by sampling with different accuracy for the three power plant example introduced in Section 10.5. Figures adapted from (Schiendorfer, 2013).

$$\forall t \in \mathcal{T} : \mathcal{P}^v_{t+1} \leq \mathcal{P}^{v+}_{\delta}(\mathcal{P}^v_t)$$
$$\forall t \in \mathcal{T} : \mathcal{P}^v_{t+1} \geq \mathcal{P}^{v-}_{\delta}(\mathcal{P}^v_t)$$

Figure 11.6 provides an additional example of the sampling of the cost function.

As Algorithm 3 potentially generates a large number of possibly NP-hard problems, we need to make sure that the time spent for sampling is bounded, e.g., by setting a time limit and giving up on an input point if no solution is found after that limit. Anytime algorithms can also be used for sampling runs with a fixed duration. If further properties of the function are known such as monotonicity ($x \leq y \rightarrow f(x) \leq f(y)$) or extensivity ($x \leq f(x)$) this information can help to shrink the search space for the resulting sampling problems by introducing constraints that enforce these properties. Table 11.3 indicates that using more sampling points leads to increased accuracy in abstraction. In practice, an incremental approach that samples the models in the background is useful to obtain more accurate representations while initially offering a crude version of the PWLs.

Reformulating and Solving SRMs for Sampling

The constraints available in the SRM are used to formulate new problems suitable for sampling. Compared to the standard synthesised regional model, no residual load or initial states are included in these "sampling" models. A special role is given to the decision variables for the agents' production at time step 0. Usually constraints link the current state to these variables, e.g. $\mathcal{P}^a_0 = 10.0$. But in this case these constraints are dropped and an additional constraint only ensures that a suitable initial state is found which is bound to an aggregate value s that corresponds to the sampling point: $\sum_{a \in \mathcal{A}} \mathcal{P}^a_0 = s$. Then a suitable valuation for the productions at 0 is calculated by the solver and a target expression is maximized or minimized as outlined above.

Sampling abstraction constitutes an approximation of functional relationships, even in cases where temporal abstraction can not be applied due to the specific formulation of the constraints. However, since it is based on the repeated solution of the overall problem, it incurs a performance penalty that might play a role in some scenarios. In general, solving the optimisation problems for sampling should be much faster than solving a synthesised model at the higher level. As it is not necessary to find the *optimal* solution to the sampled problems but rather an approximation is required, the use of an anytime algorithm that can be terminated at any point is time is advised. This allows setting a time out after which a *feasible* solution will be returned. If no solution has been found in the allotted time frame, interpolation between sampling points can be used. A further technique that can be applied is the successive refinement of the abstraction at runtime. The performance of the abstraction is mainly important after a VA has been altered or newly formed. In such a case, it is vital that an abstraction is available quickly. Once this has been communicated to the superordinate VA, sampling can continue to run, refining the abstraction by either sampling at additional points or by looking for better solutions at existing sampling points. A better solution can then be communicated as soon as it becomes available.

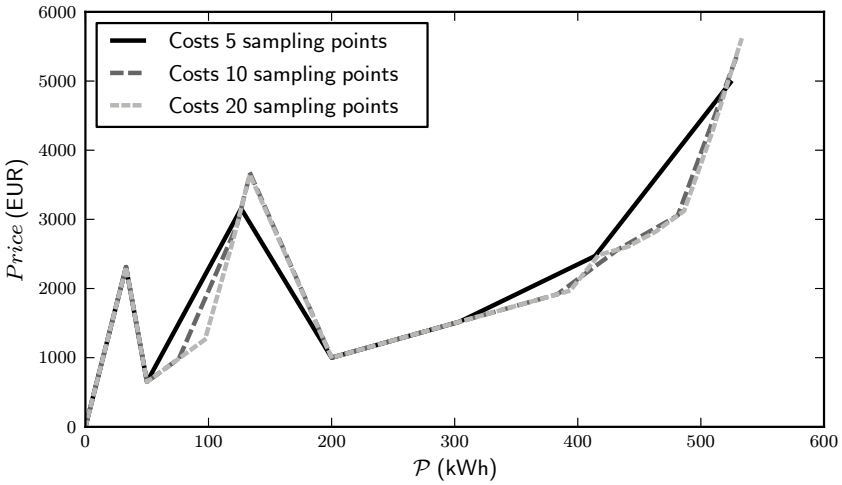


Figure 11.6: Cost function sampled at different accuracies for the virtual agent w consisting of three concrete agents: **b1**: $L^b1 = \langle[0,0],[50,100]\rangle$, $\gamma^b1 = 13$; **b2**: $L^b2 = \langle[0,0],[15,35]\rangle$, $\gamma^b2 = 70$; **w1**: $L^w1 = \langle[0,0],[200,400]\rangle$, $\gamma^w1 = 5$; where γ is the price per production unit such that the total cost for one time step is given by $\sum_{a\in\{b1,b2,w1\}} \gamma^a \mathcal{P}_t^a$. General abstraction gave the feasible production space of this virtual agent as $L^w = \langle[0.0,0.0],[15.0,35.0],[50.0,135.0],[200.0,535.0]\rangle$. Different numbers of sampling points are selected equidistantly over the whole production space in addition to the boundary points of the intervals. Therefore, the more sampling points are used the higher the accuracy of the obtained piecewise linear function. The objective was to minimize the cost given a combined production. Low production rates are quite expensive. Figure adapted from (Schiendorfer, 2013).

11.4 Abstraction of Optimisation Criteria given by Constraint Relationships

Sampling point approximation can also be used to abstract the optimisation criteria provided by constraint relationships in the synthesised models. In fact, the penalties incurred by violating soft constraints depend solely on the assignment to the solution variables. For a given solution, it is thus possible to calculate the minimal sum of penalties incurred by this solution if the underlying model is set to minimise the penalties. When performing this solution process for a number of feasible solutions, the penalties incurred can be expressed as a piecewise linear function just as described in Section 11.3.

In general, sampling penalties follows the same principle as outlined above. Soft constraints limiting the rate of change, e.g., depend on two time steps. In the sampling used so far, the production for the second time step was calculated while an objective function representing the functional relationship of interest was minimised or maximised. Now, however, we are interested in the penalty for a specific future development of the system. Therefore, it is necessary to introduce two constraints $\sum_{a\in\mathcal{A}} \mathcal{P}_0^a = x$ and $\sum_{a\in\mathcal{A}} \mathcal{P}_1^a = s$ where the former constraint defines the production in time step x and the latter is the sampling point s.

Using only an objective function that minimises the penalty, it is now possible to determine which penalty is at least incurred by a specific solution. Maximisation is not necessary in this case since it is always the goal to minimise penalties. However, if the soft constraints are formulated over a longer time span than two time steps—e.g., a MaxSPANconstraint that limits load changes such as the one introduced in Section 9.2—this approach is insufficient. In such cases, a complete schedule would have to be evaluated against the soft constraints. This is, however, not practical due to the combinatorial explosion of possible sampling points. We therefore focus on one-dimensional (e.g., for economic optimisation) and two-dimensional (e.g., for rates of change) problems and ignore higher-dimensional ones as abstraction artefacts for the moment.

Figure 11.7 shows the piecewise linear function for the one-dimensional soft constraints in the constraint relationship graph depicted in Figure 10.3. The graph also shows that a coarse-grained resolution for the sampling introduces abstraction errors, e.g., when an output of 416 kw is sampled only if 20 sampling points are present. In addition to the sampling points, the boundaries of the feasible ranges are always sampled as well. Therefore, an experiment with 5 sampling points can actually have more than 5 points—in the case depicted in Figure 11.7, there are 9 sampling points. In general, using a

higher number of sampling points does not necessarily yield better results as discussed in Section 11.5.

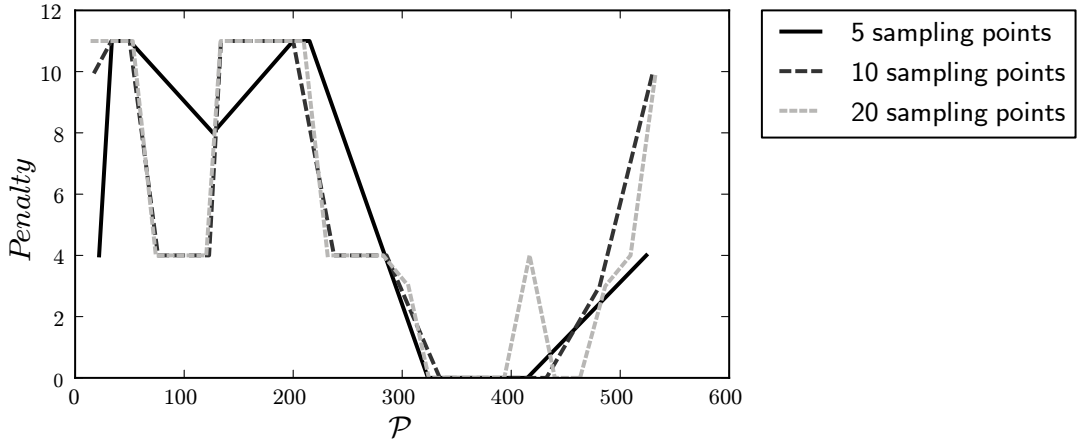


Figure 11.7: The piecewise linear functions for the penalties of the synthesised model in Figure 10.3 considering the constraints for economic optimisation for different numbers of sampling points. The slopes between the sampling points are linear approximations. Please note that the three experiments in general do not share sampling points. This becomes evident in the last sampling point that is different in all three scenarios due to the flexibility of the input values (524.0, 530.11, 532.68). It becomes clear that the higher resolutions yield more accurate results, especially in cases that are missed by the lower resolution experiments (e.g., 416.90 is only considered in the experiment with 20 sampling points).

Different target productions yield different overall penalties. Incorporating the constraints that limit the rate of change of the power plants introduces an additional dimension that can be visualised as an additional axis. Once the penalties have been captured as a PWL, they can be made available in the abstract agent model of an intermediary w and consequently in the synthesised regional model containing this AAM by assigning the value of the function to one of the slots of the penalty array.

```
penalties["w"][t] == w_penaltyfunction(production[w][t]);
```

11.5 Abstraction Errors and Runtime of the Abstraction Process

The main evaluation criteria to determine the efficiency and usefulness of the proposed abstraction mechanism are solution time and deviation from non-abstracted solutions. These criteria have been assessed in the context of the autonomous power management case study in (Schiendorfer, 2013). The results shown hereafter have been adapted from this source. It used a centralised, non-abstracted model to create globally optimal solutions to the scheduling problem and an agent-based regio-central solution process that incorporates model abstraction.

Design of the Experiments

As in the rest of this work, power plant models are formulated in OPL, the input language of IBM ILOG CPLEX (CPLEX, 2013). Consequently, they are defined as mixed-integer programs. The central and regio-central models are both solved using the same system configuration and initial state as well as the same development of the system over time with regard to demands and intermittent power production. An experiment run is defined by the following parameters:

n: the number of power plants.

s: the number of sampling points.

hc: the hierarchy creation strategy. This strategy defines the initial structure of the system and influences both the composition of the AVPPs and the height h of the resulting tree. It is either possible to define the maximum number of children per AVPP (thus, a single parameter is used for this strategy) or to define the number of physical power plants that should be controlled by "leaf" AVPPs and the number of AVPPs controlled by an AVPP within the hierarchy (thus, two

parameters are used). The latter strategy is analogous to a B+ tree. With regard to the HiSPADA self-organisation approach (cf. Chapter 4), both kinds of structures can be produced, depending on the constraints used in decision making. The parametrisation used in Chapter 4, however, produces AVPPs of the former kind which are therefore also the focus of the evaluation.

The plant models are also somewhat flexible in the sense that the constraints for a specific plant are randomly chosen from a set of constraints so that reasonable power plant models are created. These constraints define minimal and maximal production boundaries and change speeds. These non-deterministic aspect, as well as the initial states of the power plants and the hierarchical structure, are controlled by random seeds which are also part of the specification of the experiment. Together with the parameters introduced above, the seed allows to repeat individual experiment runs. As in Section 9.3, realistic production boundaries and load curves are used (cf. p. 142).

Each simulation run covers half a day, during which 43 time steps that each cover 15 minutes are performed. In each time step, a schedule is calculated for 5 time steps in advance. The centralised model is solved by a single AVPP that contains all physical power plants, thus using exactly the same infrastructure and solver configuration as the regio-central approach.

A number of criteria are used that are indicative of the quality of the solutions as listed below. The subscripts c and rc indicate central and regio-central, respectively. All values in the tables showing the results are averaged over multiple runs with different random seeds.

tr: the *t*otal *r*untime for the top-level resource allocation problem.

$\mathbf{tr_{sch}}$/t: the *r*untime per *t*ime step used only for the solution of the *scheduling* problem. For central solutions, this is equivalent to tr divided by the number of time steps for which scheduling is performed (usually less than the total number of steps in the experiment since scheduling is performed for multiple time steps in advance). For regio-central solutions, this number corresponds to tr_{rc} minus time required for abstraction.

$\mathbf{tr}_{abs}$: the *t*otal *r*untime spent for *abs*traction. Since general and sampling abstraction are only performed at the beginning of the experiment and temporal abstraction occurs in every time step, the different abstractions are categorised as "fixed" (tr^f_{abs}) and "variable" (tr^v_{abs}) runtimes.

v: the total de*v*iation between demand and production in relation to the total demand;

ae: the *a*bstraction *e*rror resulting from an assignment that can not in fact be fulfilled by an AVPP but that the abstraction claims it can. This error is given as a relative number based on the assigned production. If an AVPP assigns a production of 100kW, e.g., and the AVPP is only able to provide 95kW, the abstraction error is 5%.

The experiments were run on an 8-core Intel Xeon 3.20 GHz machine with 14.7 GB of RAM. A 64 bit Windows 7 OS with 8 GB RAM available to an Oracle Java 7 JVM was used. The abstraction algorithm as well as the CPLEX optimizer executed in the same JVM. Time limits were imposed on the calculation time for each time step as a realistic limit to how fast the schedule needs to be available to be able to react to changes in it. To give the solver more leeway, 30 minutes were used instead of 15 minutes required in the real world. This compromise was made to ensure that the central solutions give a high-quality baseline for comparison with the regio-central approach. Since CPLEX uses a branch-and-bound algorithm that is able to return a result as soon as one was found but does not necessarily return the best result if stopped before the tree could be searched exhaustively, there is no guarantee that the central solution is optimal. If the time limit was violated before a result could be found, the central solver was stopped. Since no randomized variables were used in the experiments, single representative runs were selected for the presentation of the results.

Experimental Results

To accommodate the different evaluation areas of performance and accuracy, three questions are distinguished: scalability issues and the changes of the solution quality for different problem sizes, the influence of hierarchical system structures, and the influence of the number of sampling points on quality and performance.

Scalability and Solution Quality: *How does the size of the problem impact performance in terms of time and quality?* We expect that for systems of a certain size, the additional time spent on the abstraction process outweighs the runtime of a central solution. Furthermore, we expect the regio-central approach with its abstracted models to exhibit slightly higher deviations while still being within

n	50	100	500	1000
tr_c	1995.29	2231.21	40850.04	51557.12
tr_c/t	46.40	51.89	950.00	1199.00
v_c	1.26%	0.36%	3.20%	3.30%
tr_{rc}	318.50	1036.22	8073.79	16018.25
tr_{rc}/t	4.50	18.85	139.79	311.43
v_{rc}	**1.30%**	**0.61%**	**1.30%**	**2.20%**
tr_{rc}/tr_c	15.96%	46.4%	19.76%	31.07%
h	1	1	2	3
tr^f_{abs}	125.10	224.86	2079.21	2626.7
tr^v_{abs}	0.28	1.00	17.00	32.00
tr^v_{abs}/t	0.007	0.024	0.39	0.73
ae	**0.0079%**	**0.2%**	**0.2%**	**0.7%**

Table 11.1: Comparison of measurements depending on different power plant numbers. Values below the horizontal line are only relevant to the regio-central approach. Times are given in seconds and c and rc subscripts denote central and regio-central, respectively.

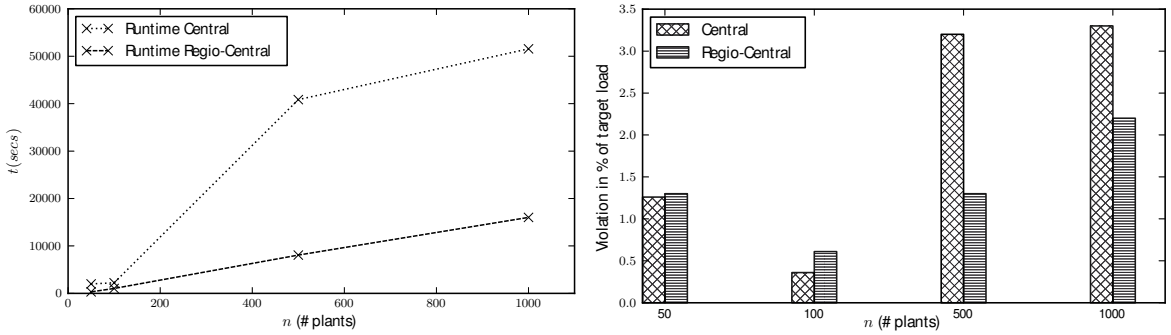
acceptable bounds. To be able to compare the results, a fixed hierarchical structure is used in which leaf AVPPs subsume 20 physical power plants each and inner AVPPs in turn control 5 AVPPs each. The hierarchy thus constitutes a form of "B+ tree" Experiments with more power plants therefore use a deeper hierarchy. The scalability and the solution quality of the regio-central approach is illustrated in Table 11.1 for small to large systems. Overall, the results support the claim that a regio-central approach using abstraction is suitable for practical settings in distributed energy management.

The important measures to compare with regard to the *solution time* are tr_c/t and tr_{rc}/t, the averaged runtimes per step for the *c*entral and for the *r*egio-*c*entral approach. The runtime per time step is highly relevant for the continuous operation of the control scheme since scheduling is repeated in all steps. The regio-central approach performs very well in this case. The average times required for abstraction tr^v_{abs} remain below one minute even for 1000 power plants. However, the initial creation of the abstracted models can incur a slightly higher runtime in the beginning of the runs. The relation of the total runtimes for solving the scheduling problem in both approaches is indicated in row tr_{rc}/tr_c and clearly shows that the regio-central approach outperforms the central one in all problem instances. These total runtimes are also compared in Figure 11.8a.

The average violation, i.e., the deviation between demand and production remains (v_c and v_{rc} for central and regio-central solutions, respectively) are very close to each other. For larger system sizes, the regio-central method even performed better than the central model as shown in the table and in Figure 11.8b. While counter-intuitive at first sight, this phenomenon can be explained by the cut-off in scheduling time: if the solver can not find a solution within 30 minutes, the power plants will operate with their previous schedule, not taking into account new information. Since this effect is mainly visible in large systems due to the increased search space, these scenarios are vulnerable to such effects. Figure 11.9 shows the total violation for 1000 power plants as indicated in Table 11.1.

Hierarchy influence: *How does the depth of the hierarchy affect solution quality and runtime?* To evaluate the influence of the structure of the hierarchy to runtime and solution quality, the number of power plants per AVPP is varied. This changes the complexity of abstraction and introduces the potential of abstraction errors. In addition, as the overall runtime depends on the number of solutions that have to be calculated and on the individual problem size (see the discussion on maximal sequential runtime of scheduling in Section 4.5), these variations give an indication of the changes in runtime for different hierarchical structures. Table 11.2 summarises the results for different input sizes n and plants per AVPP p (the capacity).

The hierarchy in experiments with 500 power plants is inclined towards the leaves, meaning that physical power plants are mainly assigned to AVPPs on the lowest level. This is a realistic assumption since most smaller power plants will be located in the lower parts of the hierarchy, especially when considering the different voltage levels of power grids. In addition, this incurs performance benefits since the largest portion of controllable power plants are abstracted at a rather low level so that higher



(a) Total running times central vs. regio-central.

(b) Violation grouped by problem size and algorithm.

Figure 11.8: Illustration of the scalability and solution quality of model abstraction in the power management case study. The figure on the left-hand side shows the respective runtimes, the figure on the right-hand side shows a comparison between the difference in demand and production for the central and the regio-central case. For large problems, both runtime and violation decrease, showing the viability of the approach and its benefits for large-scale systems.

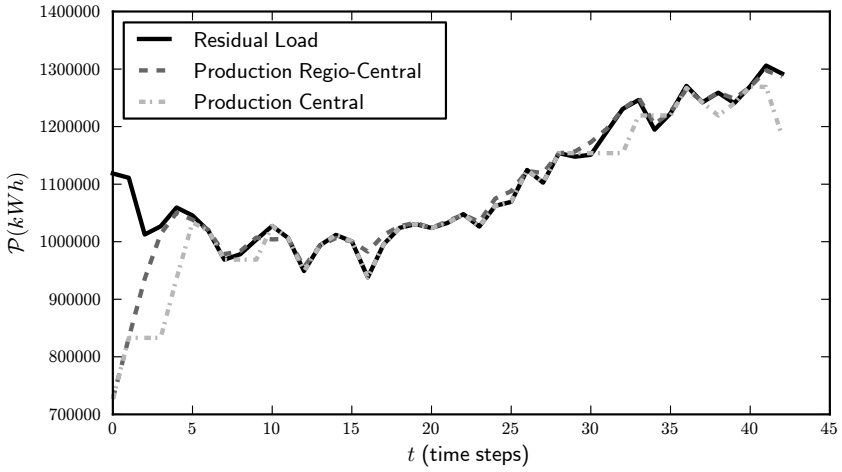


Figure 11.9: Solution for 1000 power plants and a half day in 15 minute steps, central, taken from Table 11.1; tableaus in the central model indicate that no solution was found after 30 minutes.

levels have to solve smaller problems with less variables.

The results show that the capacity of the AVPPs affects quality, abstraction error, and runtime of the runs. In general, the number of power plants per AVPP greatly influences solution quality in the AVPP as more information and especially more degrees of freedom become available to solve the local problem. The abstraction error increases mainly due to the deeper hierarchy and the additional abstraction steps this incurs. These advantages of larger AVPPs easily outweigh the added runtime that is noticeable in larger systems. Small AVPPs make for a deeper hierarchy that causes the average run time per time step to increase as compared to larger AVPPs in the experiment with 100 power plants. In the experiment with 500 power plants, however, there are more AVPPs in the high and mid-level of the hierarchy, all solving more complex problems than in the case with small AVPPs. Therefore, the overall runtime increases. However, since violation and abstraction error decrease at the same time, the added runtime might be acceptable.

Sampling Accuracy: *How many sampling points are needed to guarantee good accuracy while keeping time spent on abstraction limited?* We expected a trade-off between the time spent on abstraction, a factor mainly influenced by the number of sampling points used in the abstraction of non-linear process, and the accuracy that can be achieved as measured by the abstraction error. To determine this trade-off, experiments with 100 power plants, 48 time steps, and a varying number of sampling points have been performed. The results are given in Table 11.3.

(n, p)	$100, 5$	$100, 15$	$500, 5$	$500, 15$
tr_c	2389.31	2389.31	67403.25	67403.25
tr_c/t	55.56	55.56	1567.51	1567.51
v_c	0.36%	0.36%	3.80%	3.80%
tr_{rc}	1261.49	821.00	3820.03	6832.20
tr_{rc}/t	**7.96**	**6.38**	**39.80**	**104.41**
v_{rc}	**0.97%**	**0.51%**	**1.80%**	**1.62%**
tr^f_{abs}	919.28	546.75	2108.64	2342.92
tr^v_{abs}/t	0.005	0.007	0.008	0.057
ae	**3.80%**	**0.3%**	**2.10%**	**0.49%**

Table 11.2: Comparison of different hierarchies for different problem sizes. As the central problem is independent of AVPP capacity, the runtime of the central solution does not change for different p.

The violation in the central solution of the problem v_c is 0.36% and the overall solution time is 2235.83 seconds. As indicated by the previous experiments, none of the runs using abstraction and regio-central solutions are able to achieve the same accuracy, but the runtime is very competitive. The run using 20 sampling points achieves the best accuracy with a violation of only 0.45%, but the overall runtime is higher than in the central case due to the time spent with the initial abstraction. Depending on the accuracy required, 5 or 10 sampling points approximate the models used in the experiments quite well. The 10 sampling point run also has a very low abstraction error. As a caveat, these observations are not readily generalisable due to the specific way power plants and especially the rate of change are modelled and behave.

It is interesting to note that the abstraction error is slightly worse for the experiments with higher sampling frequencies. This is due to the flexibility in the sampling points described in Section 11.3 that allows to explore the search space on the one hand and yields solutions more quickly on the other hand. This flexibility implies, however, that the sampling points for the same base data but for different sampling point numbers are not equal. Slight aberrations in the concrete sampling points can, however, yield slightly different results. This is especially evident when sampling the penalties of a synthesised model as described in Section 11.4. If the flexibility allows for a deviation of up to 2%, a sampling point that was originally 530kW can be changed to one at 524kW. In that case, the lower output can allow finding a schedule in which one of the power plants runs at its economical optimum, thus incurring a much lower penalty than with the original output.

The abstraction error depicted in Table 11.3 compares the results of a hierarchical model abstraction process with an optimal solution found with a centralised model without regard of penalties. In such a case, the additional sampling points can potentially have the same effect of adding information to the abstracted model that is not absolutely precise. Sine the abstraction errors are very low, however, this additional uncertainty can be neglected in the case study. For other specific applications, a similar analysis has to be performed prior to using flexible sampling points, however.

11.6 Model Abstraction in the Literature

Hierarchical optimisation has also been employed in the field of artificial intelligence. For example, costly automated planning routines can be pruned early if higher levels of a hierarchy reflect the adherence to a plan's most critical variables (Sacerdoti, 1974). Similarly, reflective agents need to plan their coordination—hierarchical abstractions of their interaction partners may increase their decision performance, too (Durfee, 1999).

The term "model abstraction" was coined by the simulation engineering community and Frantz (1995) provides a taxonomy of common model abstraction techniques, some of which we use in our approach as well. As an example of work in this area, Lee and Fishwick (1996) used neural networks to get a behavioural abstraction of subcomponents that were given as state machines. Hierarchical decomposition has been found useful in genetic algorithms (Pelikan and Goldberg, 2000) by combining solutions from lower levels to solutions on higher levels similar to the presented method. While they also made a case for hierarchical problem solving, they put emphasis on how to improve existing genetic and evolutionary algorithms. Our methods works with any existing constraint optimization algorithms but

s	5	10	15	20
tr_c	2235.83	2235.83	2235.83	2235.83
tr_c/t	52.00	52.00	52.00	52.00
v_c	0.36%	0.36%	0.36%	0.36%
tr_{rc}	1036.22	2104.77	2035.38	3342.05
tr_{rc}/t	18.85	29.34	17.26	35.26
v_{rc}	**0.61%**	**0.49%**	**0.49%**	**0.45%**
tr^f_{abs}	224.86	842.99	1293.40	1825.62
tr^v_{abs}	1.00	7.00	18.00	15.00
tr^v_{abs}/t	0.024	0.17	0.42	0.34
ae	**0.2%**	**0.13%**	**0.15%**	**0.14%**

Table 11.3: Comparison of different sampling frequencies. As the problem is equivalent for all runs and thus objectives are equal for all runs, the runtimes of one central solutions are shown here as a representative sample.

is designed for a particular problem class. Kinnebrew and Biswas (2009) developed a hierarchical variant of the contract net protocol that also offered scalability benefits. Choueiry (1994) presented abstraction methods for task and resource allocation problems and focused on heuristics that find interchangeable sets of tasks. The approach presented here depends on an interval representation and does not cover tasks that share resources but rather concentrates on resource allocation problems without side effects as defined in Section 10.1.

However, the model synthesis and abstraction can be construed as an instance of self-organised middle-out learning (SOMO, cf. Mammen et al., 2011; von Mammen and Steghöfer, 2013, 2014). SOMO is an approach that automatically builds abstractions of processes occurring in large-scale systems from the bottom up and validates and revokes them top-down. As it works in both directions and as it bridges the gap between the levels of observation of the model, it can be considered to operate at the meso-level of analysis and thus "from the middle out". It automatically identifies process patterns—basically repeatedly occurring interactions and state changes. Instead of considering the series of all conditions that lead to the process' changes one step at a time, it suffices to recognise the occurrence of the pattern. As a consequence, the detailed interactions are no longer executed but, whenever the according preconditions hold, the observed side effects are enacted in the system. Such automatically learned patterns may also be understood as abstracted process descriptions. Abstracted processes in turn participate in interactions and can thus be further abstracted, forming a hierarchical structure of abstraction similar to the one envisioned here.

Chapter Summary and Outlook

The sampling techniques discussed in this chapter provide a way to abstract synthesised agent models to make them available in a hierarchical solution approach to resource allocation problems. The approaches outlined here are aimed at identifying the distinguishing features of a model while keeping the time required for abstraction as well as the abstraction error low. They make use of assumptions about the structure of the models that are general enough to be applicable to resource allocation problems in general. Time-dependent and non-linear behaviour can be abstracted as well. Together with the general approaches for constraint satisfaction and optimisation problems in adaptive systems introduced in Chapter 9 and the synthesis techniques detailed in Chapter 10, the abstraction of models allows the effective use of regio-central constraint optimisation approaches in large-scale hierarchical systems.

Part V

Agent-oriented Software Engineering for Large-Scale Open Self-Organising Systems

Custom AOSE methodology as the framework in which techniques developed in the thesis are embedded, amendment with supporting guidelines, and customisation to project requirements.

Chapter 12

Existing Methodologies and Requirements for an AOSE Methodology for Large-Scale Open Self-Organising Systems

Summary. This chapter gives an overview of the requirements a software engineering process for open, heterogeneous self-organising systems has to fulfil. In particular, an emphasis is placed on the interaction between agents, the independence from a concrete agent model, implementation framework, or toolkit, and the coverage of the full life cycle of a system, including its deployment and subsequent maintenance. The most significant existing agent-oriented software engineering methodologies are introduced and compared with regard to the specified requirements.

Previous chapters have introduced a number of novel approaches to deal with scale, openness, heterogeneity, and self-organisation in complex multi-agent systems. These approaches have incorporated methods to ensure the correctness of adaptation processes in an attempt to improve the trustworthiness of the systems they are applied in at runtime. Overall system trustworthiness is only ensured, however, if there is a design process that is clearly defined, documents the steps taken, and enables the designers to make conscious decisions about all aspects that are relevant to the robustness, correctness, and solution quality of the system under development. Such a process also serves as an operationalisation of the solutions that have been developed to deal with the specific characteristics inherent in the class of systems we are concerned with.

This chapter introduces the foundations of process engineering and method engineering for multi-agent systems used in the development of a methodology for large-scale open self-organising systems in Section 12.1. It outlines the current research landscape in the field of agent-oriented software engineering and important topics that influence the creation of novel methodologies in this area in Section 12.2. Section 12.3 then lists the requirements a methodology for the system class considered in this thesis has to fulfil. These requirements are discussed in the context of existing methodologies in Section 12.4, along with a discussion of the drawbacks of existing methodologies and the difficulties in evaluating them. These findings have been used in the development of the **P**rocess for **o**pen, **s**elf-**o**rganising **M**ulti-**A**gent **S**ystems (PosoMAS) which will be introduced in the next chapter.

12.1 Overview of Process Engineering and Method Engineering

A *software engineering process* defines the concrete course of action for the development of a specific software product. The term *methodology*, an umbrella term for all concepts, models, directives, and guidelines that can be used for the development of a software product, is sometimes used as a synonym for process. We will, however, adopt the definitions put forward by Henderson-Sellers and Ralyté (2010), where a methodology is a synonym of *method* and denotes "an approach to perform a software/systems development project, based on a specific way of thinking, consisting, inter alia, of guidelines, rules and heuristics, structured systematically in terms of development activities, with corresponding development work products and developer roles". A methodology thus contains one or more processes, i.e., steps that have to be taken in the creation of a software product (the dynamic aspects, so to speak), aspects

of the product, i.e., the tangible artefacts that are created during the process (the static aspect), and the people that perform the process (the human aspect). Concretely, this includes the artefacts that have to be created, a number of activities and their preferred sequence, stakeholders that have to be involved, notations, and more.

Situational Method Engineering

The methodological foundations for the creation of new software engineering methodologies are *process engineering* and *method engineering*. Process engineering can be defined as the "study, evaluation and adaptation of SE processes" (Marttiin and Koskinen, 1998). More recent definitions stress compositionality and reuse. Cossentino et al. (2011) define the term as the "analysis, decomposition, and creation of software engineering processes". Method engineering, on the other hand, emphasises the combination of existing method fragments or other reusable assets to a software engineering methodology according to requirements for a software project or a class of projects (Low et al., 2009; Xiao and XueYan, 2008). **Situational method engineering** (Henderson-Sellers and Ralyté, 2010) is used if the focus is on composition of new, ad-hoc methodologies for specific projects or development environments. The distinction becomes clear in the light of the definitions above: method engineering is focused on the combination of individual parts of a methodology and the three different aspects while process engineering is concerned with the course of action. However, many authors make no implicit distinction between these two terms, often using them as synonyms (Cossentino et al., 2011). We will use the term situational method engineering (SME) from here on out to denote the creation of a customised software engineering methodology from reusable assets that contains the dynamic, the static, and the human aspect.

Reusable Assets

To enable effective SME, a collection of reusable assets has to be available. These have been identified before by analysing existing processes or formally describing building blocks of processes. They are collected in a *method library* or *method repository* (Henderson-Sellers et al., 2008). The form these assets take can differ. The literature distinguishes *method fragments* and *method chunks*.

A **method fragment** is generally considered the smaller building block and can be defined as "a piece of an information systems development process" according to Cossentino et al. (2011). Following (Brinkkemper et al., 1999), method fragments can be differentiated by the aspect they cover: product fragments represent individual artefacts while process fragments represent stages, activities and tasks. Henderson-Sellers et al. (2008) agree, and add that a method fragment is instantiated from a metamodel for processes and usually includes an association between product and process fragments. We use the term to denote a part of the methodology executed while going through the process to achieve a certain aim.

In contrast, a **method chunk** is a "consistent and autonomous component of a development process" (Cossentino et al., 2011). More specifically, it is a combination of a process and a product part (Ralyté and Rolland, 2001) with an additional interface describing the methodological situation it can be applied in and the objective it allows to be achieved as well as a descriptor that contains meta-data and reuse information (Henderson-Sellers et al., 2008).

The reusable assets that will be used in the following adhere to the concepts of a **practice**, defined as "a documented approach to solving one or several commonly occurring problems"[1]. The concept is not generally found in the literature but is used extensively as building blocks for highly customisable methodologies within the Eclipse Process Framework (EPF)[2] and within the IBM Rational software suite[3]. Practices are close to method chunks, but can contain several fragments of the different aspects and are thus more coarse-grained. On the one hand, this makes it easier to compose methodologies as less practices have to be selected and less alternatives have to be considered. On the other hand, the combination of chunks or fragments allows more fine-grained customisation. The latter point is addressed with several measures to increase customisability, e.g., the use of "work product slots" that allow exchanging artefacts (e.g., use case models) with equivalent but different ones (e.g., goal-based

[1] http://epf.eclipse.org/wikis/epfpractices/practice.bus.mdev.base/guidances/concepts/practice_ACB683BB.html?nodeId=dc9a7020

[2] http://epf.eclipse.org

[3] http://www.ibm.com/developerworks/rational/practices/

requirements models). This way, practices become interchangeable and can be freely combined. Preconditions, postconditions, inputs and outputs are clearly defined and thus allow directed composition as well as consistency checking.

Reusable assets are collected in *repositories* where they are stored in a form that allows easy access and reuse. If such repositories existed, a method engineer could draw from them, select the adequate asset based on the selected criteria, and adapt them so they fit together and form a complete methodology. This ideal of method engineering is the reason for repeated calls for such repositories for agent-oriented methodologies, e.g. in (Cossentino et al., 2011). Indeed, some efforts have been made by these authors to provide a repository (Seidita et al., 2006), but the result[4] contains only fragments from the PASSI methodology described as text along with some diagrams. As of August 2013, the repository website claims that it is only intended as a proof of an ongoing discussion within the IEEE FIPA Design Process Documentation and Fragmentation Working Group. Seidita et al. (2006) additionally mention the method base that has been created as part of the Decamerone tool (Harmsen and Brinkkemper, 1995) proposed in 1995 and no longer available online, the method base of the aforementioned FIPA committee which has not been completed yet, and the fragment archive that is part of the Open Process Framework (OPF, see sidebar on page 179).

The Role of Meta-Models in Method Engineering

Practices can be defined in the Eclipse Process Framework Composer. This tool allows to model process elements based on SPEM, the "Software & Systems Process Engineering Meta-Model" (OMG, 2008). It defines the concepts necessary for this purpose, including process behaviour (dynamic aspect), method content (static aspect), and their relationship. A graphical notation—introduced in the box on page 191—is available for the different elements of the meta-model. When defining process elements in the Eclipse Process Framework, the modeller instantiates the concepts defined in the meta-model. For instance, the modeller instantiates the concept `TaskDefinition` to describe a job that has to be performed. The meta-model prescribes that a task has `Performers` and `Inputs` as well as `Outputs` and consists of several `Steps`. A task definition is an abstract description of what the task entails. To denote that a task is performed at a certain point in a process, a `TaskUse` is defined within an `Activity`, i.e., a sequence of task uses.

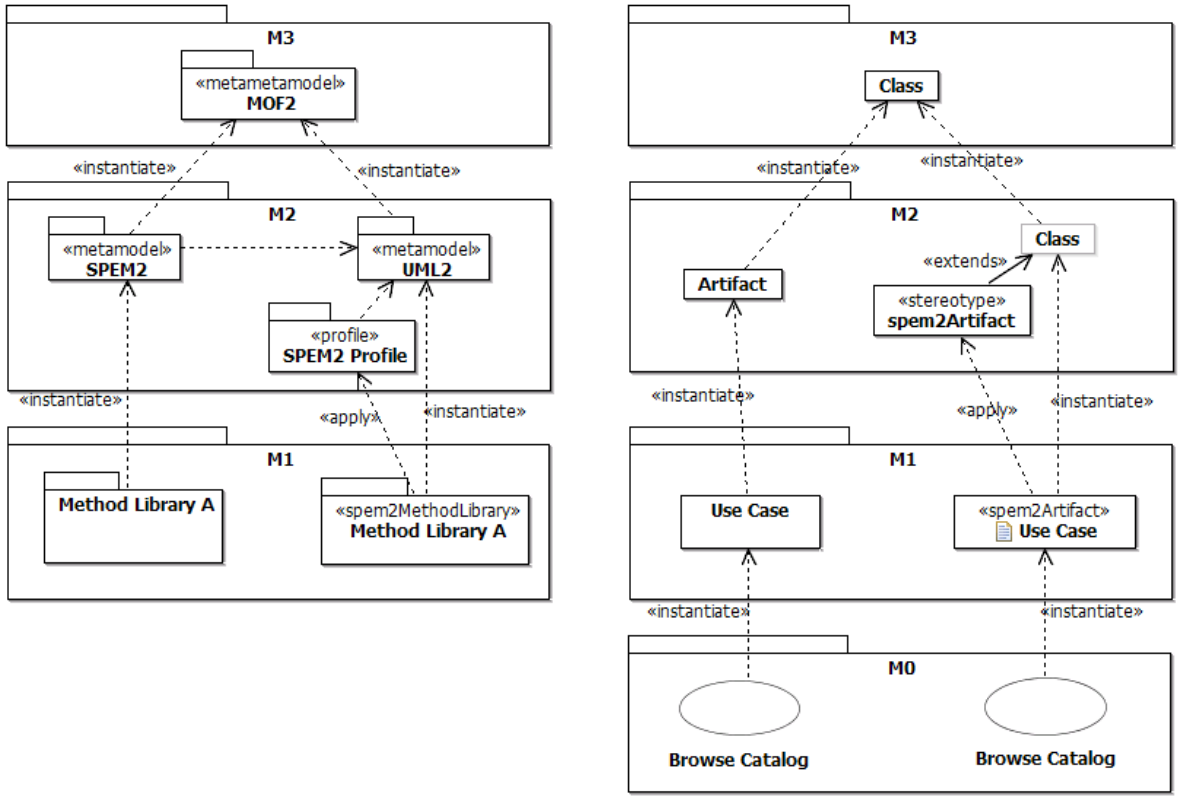
In other words, the model defined on the higher layer defines the language to be used on the layer below (OMG, 2008). Thus, SPEM is the meta-model (model layer M2) that describes the concepts used in the definition of a process. The process itself (sometimes also called "process model", cf. Henderson-Sellers and Ralyté, 2010), modelled in the concepts defined by SPEM, is the model on model layer M1 that describes the concepts used in the instantiation of the process. Any process customisation is also done on this layer. Layer M1 defines the language on which the process instantiation located on model layer M0 is based and according to which the developers work. These relationships are illustrated in Figure 12.1. The strict adherence to a meta-model allows the method engineer to validate the process by checking whether the concepts defined in the process fit (e.g., the inputs and outputs of sequential activities have to match). It also clearly defines how the instantiation works and at which points extensions can be made. A tool like the EPF Composer allows to model process elements visually and save the process model in an interchangeable format. This also enables storage of the process element in a repository and easy reuse and adaptation during situational method engineering.

The OPEN Process Framework and Meta-Model

Some literature on AOSE points to the OPEN Process Framework (OPF) and uses the meta-model developed as part of this endeavour to describe method fragments. Indeed, the OPEN consortium, a group of more than 30 software engineers and researchers, provided this meta-model based on the merger of several independent methodologies as far back as the mid-nineties (Firesmith and Henderson-Sellers, 2002). In addition to the meta-model, OPF includes a rather extensive repository of method fragments (available at the project website at `http://www.opfro.org`) and a set of guidelines. It seems, however, that the framework has been discontinued. The website's last update was in 2009 and the main protagonists of the consortium have not published on OPF in several years. This might be due to the fact that SPEM has been widely adopted in the industry with powerful tools to model methodologies and that the critical mass of industrial adopters has never been achieved.

The other role of meta-models in the development of multi-agent systems is to capture the concepts required in the design of such a system. They can not only be used in the modelling of the agent architecture, but are also used in the selection of method fragments during situational method engin-

[4] `http://www.pa.icar.cnr.it/cossentino/fragrep/default.html`



(a) Abstract model layers for SPEM 2.0

(b) Example of SPEM 2.0 model layers

Figure 12.1: SPEM 2.0 is defined as a methodology meta-model (Henderson-Sellers and Ralyté, 2010) on level M2 of the meta object facility. The elements of the methodology are an instantiation of this meta-model on level M1. The concrete artefacts that are created when the methodology is applied to develop a product (the methodology instance) are on level M0. Both figures taken from (OMG, 2008).

eering. The concepts defined in the meta-model have to be instantiated in the course of applying the methodology. Thus, the selection process has to ensure that all concepts are covered by the method fragments. In theory, this would even enable automatic selection of the fragments. There are a number of meta-models for agent-based systems, but unification of these models has progressed (Beydoun et al., 2009; Low et al., 2009). However, such models are often not as generic as their authors propose. The MAS meta-model defined by Low et al. (2009), e.g., explicitly contains the BDI paradigm (see sidebar on page 181), limiting the stated generality. On the other hand, if a specific system class is addressed by a development process, a meta-model can be a valuable tool to guide the design process. The software engineering guideline proposed for self-organising resource flow systems proposed by Seebach et al. (2010), e.g., is based on a meta-model for the system class that simplifies the design tremendously by providing the necessary building blocks as a basis for custom extensions and by proposing activities based on a pre-defined structure. If a process that is based on a meta-model is used, the method engineer that selects and customises the process for the development effort has to take it into account and ensure that the system to be developed can be mapped to the meta-model.

12.2 Current trends in Agent-Oriented Software Engineering

Up until the middle of the last decade, agent-oriented software engineering was mainly concerned with closed multi-agent systems in which a number of pro-active agents collaborated to achieve a common goal. Openness, scale, and structure played a secondary role, as the authors of methodologies explored the various implications of agent-oriented design, its difference to common object-oriented approaches and the focus on agent specialities such as mental attitudes. A common example at the time, and one that still pops up even recently, was the “personal travel assistant” (see, e.g., (Núñez Suárez et al., 2000), (Bauer et al., 2001), and countless others), a system in which agents representing human users

and accessing a knowledge base of these user's preferences collaborate with agents representing airlines, hotels, car rental agencies, etc. to book the holiday of the user's dreams. The choice of this example can be better understood when we consider that the original concept of agency was conceived to express the representation of humans by autonomous processes. Even though it is debatable whether human users would ever have delegated the authority to book a complete holiday to any computer program, regardless of its sophistication, it served to illustrate vital problems in multi-agent systems: issues in the interaction of agents, including protocols, syntactic and semantic understanding, and security (O'Brien and Nicol, 1998); modelling of agent systems, especially with regard to shortcomings of UML (Bauer et al., 2001); and automatic learning of user preferences (Ng and Sussmann, 1996).

As the understanding of these factors matured (and it became more and more obvious that the personal travel assistant is not a tool that would find acceptance from a wide user-base), other aspects came into focus. The self-adaptive and self-organising systems community had adopted multi-agent systems as the ideal basis for implementations of their systems. The need to move decisions to runtime as well as for dynamic and autonomous coordination became a major concern. An early example for such adaptive systems built on agent technology were manufacturing control systems (Bussmann and Schild, 2000). The autonomy of the individual agent was also an ideal setting for systems that had grown in scale in the past years and became increasingly difficult to control manually. Such a situation has, e.g., presented itself in the case of smart grids (cf. Chapter 2). The aspects important in these settings have of course influenced the research on agent technologies and AOSE. Finally, the assumption that agents would always voluntarily cooperate to achieve a common goal (the *benevolence assumption*) has been challenged by many systems in which agents compete for scarce resources, act strategically to gain an advantage over others, or even have an incentive to cheat. An example for such an environment is agent-mediated electronic commerce (Sierra, 2004).

The BDI Paradigm

BDI, a shorthand for "belief, desire, intention" has been originally developed as a model to describe human mental attitudes and decision making. It has been adapted to agent systems in the form of *procedural reasoning* (Rao and Georgeff, 1995) in which agents select *goals* and appropriate *plans* to achieve those goals based on their *beliefs*, i.e., their internal state and the information they have collected about their environment. BDI is widely adopted as a standard agent modelling approach, but the reasoning process to select the next goal and the plans that will achieve it are usually provided by an agent framework.

Scale and self-organisation also meant a shift away from an agent-centric view of MAS towards one that is more focused on organisational issues (Ferber et al., 2004). The former view was mainly concerned with the micro-level characteristics of the individual agent, its internal architecture, its reasoning engine, its mental attitudes, knowledge representation, and its communication language. This focus has lead to a number of important results, including the standardisation of communication languages with FIPA-ACL[5] and a thorough understanding of creating agents with the BDI-paradigm. The shift towards organisations of agents has thus been a natural next step and a response to the relative maturity of the agent-centric view. In the organisational view, social aspects as well as dynamic structures formed by the agents become the focus of investigation. From a meso-level point of view, comprised of the organisations themselves, organisations interact, exchange knowledge and coordinate (Zambonelli et al., 2001). From a micro-level point of view, the individual agent now has to fill a role within one or more organisations, adhere to rules set by the organisation (Boella and Torre, 2006), and align its autonomous decisions with the aims and norms put forward by the organisation. A discussion of the consequences of these factors can be found in Chapter 3, especially with regard to organisational structures found in such systems. This shift towards organisations has also had an impact on agent-oriented software engineering, so that a "categorisation by leitmotiv" (Cossentino et al., 2010) for different paradigms—mainly by distinguishing the emphasis in the modelling process—becomes possible. More recent AOSE methodologies (e.g., O-MaSE, see Section 12.4) thus stress the organisational point of view and provide models and activities aimed at capturing organisational structure, the behaviour of organisations, and of the agent within the organisation.

One of the accompanying effects of the switch towards an organisational view is a switch towards open systems. While their characteristics are discussed in Chapter 1, it is important to notice here that openness requires all agents of the system—lest they want to risk being exploited—to adapt a strategic behaviour towards their fellows. To express this strategic behaviour, and to analyse the available alternatives for action, game theory has been widely adopted (Parsons and Wooldridge, 2002). The

[5]The Foundation for Intelligent Physical Agent (FIPA) specified a communication language for agents as a common syntactical standard in the early 2000s that is supported by many multi-agent systems such as JADE. The specification is available at `http://www.fipa.org/repository/aclspecs.html`.

interactions between agents are thus modelled as games in which it is assumed that all agents behave in a rational fashion and that the payoffs of the different outcomes of the interaction are universally known. A lot of research has been conducted to analyse agent systems with game theory and to operationalise game theoretic frameworks for decision making within agents. An alternative to the extensive use of this approach is to use trust (Steghöfer et al., 2010) to deal with unknown agent behaviour and make predictions about the potential future behaviour of the agents. Using trust values gained from the outcome of previous interactions allows the selection of trustworthy interaction partners, the formation of stable communities of agents (Bernard et al., 2012; Anders et al., 2012b), and the improvement of predictions and thus robust optimisation (Anders et al., 2013a).

12.3 Requirements for a Process for Large-Scale Open Self-Organising Systems

The main driver behind method engineering for the class of large-scale open self-organising systems is, of course, that the method must be able to deal with characteristics of the system class as outlined in Part I of this thesis. The individual requirements listed below address those characteristics and are the basis for the solution approaches that have been developed.

Accessibility to "traditional" software engineers: We strive to create a process that incorporates elements of agent-oriented software engineering approaches without alienating software engineers that have previously worked with "traditional" systems, such as service-oriented or regular object-oriented systems. One of the driving factors for this requirement is to allow designers with a regular software engineering background to use tools that they know and understand and not overwhelm them with agent-oriented specifics from the beginning. This means, e.g., that the process uses standard UML notation and thus standard modelling tools, extended with a specific UML profile. It also means that the process has to be as adaptable and customisable as standard methodologies.

Architecture and tool agnostic: The internal architecture of the agents (such as BDI) should play as little role in the high-level design part of the process as the implementation platform on which the agent system will run. Object-oriented methodologies are defined on a level of abstraction that enables this agnosticism. This does not mean, however, that these factors are completely ignored. Model-driven engineering techniques (Kent, 2002) allow the integration of these aspects later on in the system, e.g., by enabling the transformation to a platform-specific model (PSM) that contains specific model elements. Instead, the methodology should be applicable regardless of the concrete architecture and implementation platform used to allow applicability to a wide array of scenarios.

Level of detail: The process must contain support for all relevant activities in the design process. It must make clear which knowledge it assumes the designer to have and point to additional material that can be used to extend the level of understanding. The methodology must contain sufficient guidance and templates for the artefacts that must be created. The methodology should also cover the entire life cycle of a software engineering methodology, including deployment of the system.

Extensibility and customisability: The methodology must be extendible and it must be possible to combine it with different process models and to customise it for specific situations as part of a situational method engineering process. This means that it must be possible to use the method chunks put forward in an agile context (e.g., by using it in a specialised Scrum process) as well as in a heavy-weight context (e.g., using the still pervasive waterfall method), or to embed it in the risk/value life cycle provided by the OpenUP. It must also be possible to apply situational method engineering to the process to come up with a methodology suitable for the project, the team, and the environment it will be used in.

Clear separation of different architecture domains: Especially in open systems, development of the individual agents might not be in the hand of the system designer. In the energy example, the agents representing the power plants are not designed by the same people that design their interaction in the system and set up the infrastructure. Instead, the system designer has to define interfaces, data models, and interactions so that other development teams know how the agents should behave in the system, interact with other agents, and with the system as a whole. Even if the system and the individual agents are implemented by the same company (as is the case in the second case study, the Emergency Response System), different teams within the organisation can be responsible for the implementations.

Special focus on interactions between agents and agent organisations: The dynamics of an open self-organising multi-agent system are defined by the interactions between agents within organisations. The behaviour of the individual agent within an organisation plays an important role for the fitness for purpose of the organisation and of its ability to reach its goal within the system. Organisations and their structure also play an important role in terms of the scalability of the final system. Different system structures of organisations—among them hierarchical ones—must be supported by the design activities as well as requirements elicitation and analysis.

Top-down design vs. bottom-up functionality: While a systems engineering methodology is necessarily top-down, starting from overall system goals, self-organisation processes and coordinated processes within multi-agent systems provide this strived for functionality in a bottom-up way (Sudeikat et al., 2012). A methodology that is suitable for self-organising systems must take this change of perspective into account and provide appropriate tools for the design, test, and implementation of bottom-up processes.

In addition to these requirements, we adopt the principles of standard software engineering methods such as the OpenUP, that promote reuse, evolutionary design, shared vision, and others. These principles are documented, e.g., in (Kroll and Kruchten, 2003) for the Rational Unified Process, a commercial software engineering methodology that introduced many of the features that are present in modern processes such as the OpenUP. A special emphasis is put on the possibility to change the requirements at any point in time in the process. This factor is even more prominent in agile methodologies such as Scrum that will be briefly introduced in Section 13.6.

12.4 Features and Characteristics of Existing AOSE Methodologies

Scientific literature provides a plethora of AOSE methodologies, with first attempts reaching back to the 1990s. This section gives an overview of the more current approaches that have been discussed recently, pointing out their unique characteristics and shortcomings. Apart from the original papers on the methodologies, we also use content provided by attempts to compare methodologies. Such comparisons are, however, to be taken with a grain of salt, since the set of evaluation criteria used lack the same maturity the authors ascribe to the AOSE methodologies they scrutinise. In addition, some aspects of the evaluations are quite subjective. Al-Hashel et al. (2007), e.g., rate the "ease of learning" of different methodologies, without giving detailed sub-criteria. It is clear that such a criterion is very dependent on the prior knowledge of the investigator. Learning the first AOSE methodology will be much more difficult for a beginner than learning additional ones. Furthermore, some criteria are dependent on the aims of the evaluation. Abdelaziz et al. (2007), e.g., deduct points if there is no prescribed agent architecture. As explained above, we disagree and see this as an advantage due to increased flexibility. In spite of these difficulties, we apply the criteria put forward by Tran and Low (2005) for the validation of PosoMAS in relation to other processes in Chapter 14.

Another important factor is the selection of methodologies to be compared. Existing comparative literature usually selects a subset without giving specific criteria for the selection. If any are given, the selection is usually due to a perceived "maturity" of the methodologies, a factor that can hardly be measured objectively due to a lack of industrial application of the methodologies, but is rather based on the number of papers and citations and thus on the standing of the methodologies' authors in the community.

The processes selected below have been mainly chosen due to their currentness. As AOSE methodologies are usually created in an academic setting, they are developed over the course of a Ph.D. student's thesis work and disappear as soon as this student leaves the institution. Existing processes are sometimes extended (as, e.g., MasE and PASSI) to include new aspects, but often they are simply abandoned. We have chosen a number of processes which have actively been worked on in the last five years, as witnessed by publications dating from 2008 or newer. MASUP (Bastos and Blois Ribeiro, 2005), MESSAGE/UML (Caire et al., 2002) or SODA (Omicini, 2001) have been excluded for that reason. The exception to this is Prometheus, which has been included since it is the one that can be regarded the most well-studied one. Other, more recent methodologies have been excluded since the literature available about these processes does not suffice to evaluate the capabilities of the processes or their features w.r.t. the requirements outlined above. This includes AAII (Kinny et al., 1996), MASD (Abdelaziz et al., 2008), and MOBMAS (Tran and Low, 2008). Others—such as PASSI(M) (Cossentino et al., 2008)—have been excluded since the ones presented below are extensions of them.

Summary of existing AOSE methodologies[6]

The **Prometheus** methodology (Padgham and Winikoff, 2005) combines specification, design, and implementation in a detailed process and is commonly accepted as one of the most mature AOSE approaches as witnessed by the number of comparison papers that characterise it as mature and often-used (cf. Tran et al., 2005; Al-Hashel et al., 2007; Dam and Winikoff, 2003). Prometheus' aim is to provide a detailed and complete methodology for the development of multi-agent systems with BDI agents and uses a bottom-up approach. While the focus on BDI is often lauded (Al-Hashel et al., 2007; Dam and Winikoff, 2003), some authors criticise that this constitutes a restriction of application domains Prometheus is suitable for (Tran et al., 2005). On the other hand, the creators of the methodology point out that only a small subset of Prometheus is specific to BDI agents while most of it is completely independent of the concrete agent architecture (Padgham and Winikoff, 2005). A main feature are the detailed and easily comprehensible guidelines that support the different development steps, as well as support for verification, validation, code generation, consistency checks, testing and debugging. This support is provided by specialised tools and by the multi-agent platform JACK. Main points of critique are the absence of guidelines for requirements elicitation as well as the comprehensibility of the "main diagram" that combines different perspectives on the system (static, functional, and dynamic). In addition, Prometheus is unsuitable for modelling of concurrent behaviour (Al-Hashel et al., 2007).

ADELFE has been specifically developed for the design of adaptive multi-agent systems with emergent functionality. The methodology follows the Rational Unified Process (RUP) closely and uses UML and AUML[7] (Bernon et al., 2003). It follows principles from the AMAS theory as well as classical object-oriented approaches. Adherence to the AMAS theory is also the main criteria when assessing the applicability of ADELFE for a specific system: it should have a certain complexity and should be open. Additionally, the development of algorithmic solutions to the core problems is an integral part of the process and therefore, the approach is mainly suitable when the algorithms are not known yet. The agents are purely cooperative, severely limiting the notion of agent as well as its applicability for many applications where competition between agents and self-interested agents occur. An additional limiting factor is that a tool for the consistency checking of artefacts is not available (Cossentino and Seidita, 2005). ADELFE is still under development, though, and has recently been extended to provide support for adaptive agents based on learning and model-driven design (Rougemaille et al., 2009).

The recently proposed methodology **ASPECS** (*A*gent-oriented *S*oftware *P*rocess for *E*ngineering *C*omplex *S*ystems) focuses on complex systems with a particular emphasis on holonic organisations (Cossentino et al., 2010) based on the PASSI methodology. A holon is here defined as "[...] self-similar structure composed of holons as sub-structures" (see also the discussion on organisational structures in Section 3.1) and are thus very similar to the systems of systems used in the context of this thesis. The organisation into holons is captured in a meta-model that is used for the definition of the system structure. An important principle leveraged in ASPECS is the possibility of stepwise refinement of the holons. While it is possible to begin by modelling the system functionality in broad strokes, the modeller can use the nested nature of holons to refine this functionality until she arrives at the models for the individual agents. This form of decomposition lends itself to an iterative-incremental approach.

The *M*ulti*a*gent *S*ystems *E*ngineering methodology **MaSE** includes a complete development life cycle starting from the initial system specification and including implementation and deployment (DeLoach et al., 2001; Al-Hashel et al., 2007; Dam and Winikoff, 2003). It has been applied in several research projects and has been lauded for its comprehensibility (Tran et al., 2005). MaSE is independent of a specific agent architecture and can therefore be applied to heterogeneous systems (Al-Hashel et al., 2007). A strength of the methodology is the way agent interactions and protocols are defined, including a method to formally verify the correctness of protocols. Drawbacks are the difficult to understand description of concurrent tasks, the absence of models for the environment and agent behaviour, and missing specification tools for agent adaptivity (Abdelaziz et al., 2007; Dam and Winikoff, 2003). In addition, organisational factors were not considered and the methodology was very difficult to customise (DeLoach and Garcia-Ojeda, 2010). Based on this criticism, especially the last point, **O-MaSE** has been developed (DeLoach and Garcia-Ojeda, 2010). It is composed of method fragments specified as SPEM activities. The method content is based on a common meta-model and currently focuses mainly on analysis, design, and implementation. Organisations and the environment are now explicitly

[6] These summaries are based on the work of Kienberger (2012).

[7] The *A*gent *U*nified *M*odelling *L*anguage is an extension of the standard UML meta-model with agent-specific concepts (Bauer et al., 2001).

considered. O-MaSE is supported by "agentTool"[8] which provides features to graphically design the various models the fragments call for, check their consistency, and generate code for implementation platforms[9]. Since O-MaSE is a collection of method fragments, it does not in itself constitute a full process. Instead, the fragments can be combined with a number of methodologies (DeLoach and Garcia-Ojeda, 2010). In order to perform this combination, the "agentTool" provides the ability to create a custom process. For this purpose, it makes use of the modular composition of the process and provides "construction guidelines" for the combination of method fragments.

INGENIAS (Pavón and Gómez-Sanz, 2003) aims at the development of organisational multi-agent systems and is the descendant of MESSAGE (Caire et al., 2002). Just as its progenitor, it uses meta-models to describe the relevant concepts in different aspects or "viewpoints" of a MAS: organisation, describing the structure of agents, resources, goals, and tasks; agent, describing the individual agent's functionality; goals/tasks, considering their decomposition; interactions, addressing the exchange of information; and environment, defining external entities the MAS interacts with (Pavón and Gómez-Sanz, 2003). Relationships between the concepts for the different viewpoints are exploited to ensure the consistency of the modelling. Meta-models are described in a specialised modelling language. The agents are based on BDI. INGENIAS uses the life cycle of the Unified Process, allowing an iterative and incremental development. The methodology is supported by specialised tools, the "INGENIAS Development Kit (IDK)" and the "INGENIAS Editor"[10] which are still under active development. As in O-MaSE, these tools serve the dual purpose to aid in the modelling of the system to be developed and in customising the process. For this purpose, visual editors can automatically be generated from the meta-models. To customise the process, it is thus necessary to change the meta-models for the different viewpoints. The development environment also supports code generation for JADE.

Comparing and Evaluating Agent-oriented Software Engineering Approaches

It is difficult to compare AOSE processes due to a number of reasons. First of all, many of them are not universally applicable in the sense that they are focused on a specific system class. This makes it impossible to use the same case study in the comparison. Furthermore, if processes are based on specific meta-models, assumptions about the agent architecture, the implementation framework, or the resulting system design can be dramatically different. Since the outcome and the complexity of the processes are very much dependent on the design, it becomes hard to make qualitative and quantitative statements. Finally, processes are always to some degree a matter of taste and their successful execution a matter of experience. Even the selection of evaluation criteria can be subjective and based on the focus of the evaluation. For instance, Garcia et al. (2008) states that many comparisons do not take into account organisational aspects, a fact that seems natural considering that organisational issues have not been at the forefront of the respective investigations.

Many authors thus resort to comparing external properties like notation and structure of the processes or tool support (e.g., Al-Hashel et al., 2007). While these are important factors, it is doubtful that they allow comprehensive statements about the applicability and expressiveness of a software engineering process. Thus, a number of attempts have been performed to create objective evaluation frameworks that focus on the execution of the processes and use criteria rooted in the internals of the processes. Ideally, these frameworks are employed while a case study is simultaneously executed within different processes ((as done, e.g., in Abdelaziz et al., 2007; DeLoach et al., 2009). Even in such cases, however, the results of the comparison are anecdotal evidence at best and are still deeply influenced by the investigators' experience with the individual processes and the personal preferences. Furthermore, since AOSE methodologies tend to be rather specific, it is difficult to perform comparative studies by developing the same system. All applied methodologies would have to match the requirements of the system, e.g., with regard to the selected implementation platform or meta-model used.

A case-study based comparative study would, however, be the ideal evaluation: if it was possible to start a number of parallel development processes in which teams create the same system based on the same requirements, it would at least be possible to determine how well a methodology works in that specific context. The time and money required to create the project would be the final evaluation criterion. But of course, such a comparison is impossible since the application of a process is always strongly dependent on the knowledge of the software engineers participating in the project and the

[8] `http://agenttool.cis.ksu.edu/`

[9] At the time of writing, the latter feature only supports one specific simulation toolkit.

[10] `http://ingenias.sourceforge.net/`

success of a software development effort is hugely dependent on human factors (Nah et al., 2001; Cockburn and Highsmith, 2001). Since it is impossible to compose three teams of the same knowledge, dedication, and leadership, it is impossible to conduct a truly comparative study this way. Efforts that have been made in this direction have either had one team develop the same case study with several methodologies, thus neglecting learning effects and prior familiarity (e.g., by Al-Hashel et al., 2007) or by different teams, each with significant experience in the methodology applies (most prominently by DeLoach et al., 2009).

The most complete comparison of agent-oriented methodologies to date has been conducted by Tran and Low (2005). The authors use a feature analysis approach that is based on the combination of different evaluation criteria defined in previous studies, both for traditional, object-oriented approaches and for agent-oriented methodologies. The framework distinguishes four areas: process-related criteria, including aspects that are related to the process life cycle, the steps that are used, and the agent architectures that are supported; technique-related criteria, evaluating the techniques employed in the individual model steps; model-related criteria, looking at the notational aspects and the concepts included in the models used; and supportive-feature criteria, focusing on tools and support for various AOSE specialities such as mobile agents. Notably, to evaluate coverage of the development process, the authors identified 19 commonly used development steps that were confirmed to be necessary by a survey conducted among experts. While the overall evaluation suffers from some of the drawbacks pointed out above (e.g., assessment of "ease of understanding of process steps"), the feature analysis approach and the very in-depth analysis of the processes lets the study stand out in terms of quality and comprehensiveness. The findings are, however, that no single existing approach is ideal. None of them can satisfy all criteria of the catalogue. A more detailed look at some of the methodologies presented above based on Tran and Low's (2005) is given in Chapter 14—especially in relation to PosoMAS.

Disadvantages of Current Approaches

From the remarks above, it becomes clear that it is difficult to conclusively assess the capabilities of an AOSE methodology. Nevertheless, the requirements outlined above can be matched to the information that is available about the processes. As the description reveals, current AOSE methodologies do not fulfil the requirements outlined in Section 12.3 completely. They have not reached the maturity of object-oriented software engineering approaches yet and are often too specialised for a specific domain or architecture. Referring to the stated requirements, the findings can be summarised as follows:

Accessibility to "traditional" software engineers: A major drawback is the limited availability of documentation and examples. While processes such as Scrum, OpenUP, and others are well-documented and a lot of training material is available, AOSE methodologies are often only described in high-level scientific papers and thus have limited accessibility for practitioners, especially those with previous experience in traditional software engineering projects but with limited exposure to MAS concepts. This is exacerbated by the specificity of many processes (see next item) and the very abstract level of detail.

Architecture and tool agnostic: Processes such as Prometheus and INGENIAS assume a certain meta-model describing agent concepts and thus often specific agent architectures or reasoning techniques (e.g., BDI). Others, such as O-MaSE, require the designers to use a specialised software tool for modelling as they enforce custom, non-standard notation. This makes it harder to adopt the processes and creates lock-in where it is difficult to communicate the meaning of the models or switch to a different tool.

Level of detail: The lack of documentation mentioned above makes it hard to reproduce and adapt many processes. Especially the lack of method content for testing, implementation, and deployment activities makes it difficult to apply the processes in a productive setting. A notable exception is Prometheus which is lauded for its comprehensive guidelines (at least for the disciplines analysis and design).

Extensibility and customisability: Almost all existing processes can not readily be integrated with different process model (iterative, waterfall, agile) as they are not described in a modular way and thus do not lend themselves to situational method engineering approaches. More recent ones, such as ASPECS strive to remedy this situation, but currently, fragment repositories are neglected and descriptions are mainly informal. A different approach is taken by O-MaSE and INGENIAS which provide custom tools for process customisation.

Clear separation of different architecture domains (agent/system): Most of the methodologies distinguish the individual agents, the overall system, and the system environment and provide different models for each of these domains. An exception is MaSE which has been criticised for its lack of support for such specialised models, a drawback that has been fixed, however, with O-MaSE.

Special focus on interactions between agents and agent organisations: Interactions of agents are sufficiently covered by all methodologies, mostly by providing activities in the processes that deal with agent protocols and communication languages. The formation of agent organisations is treated differently, with some processes such as ASPECS emphasising their importance and providing activities and guidelines for their design, and other such as MaSE ignoring them.

Top-down design vs. bottom-up functionality: The development of algorithms is not the focus of attention of most processes. Prometheus and MaSE, e.g., do not include activities to model agent adaptivity or concurrency. Whether such activities need to be provided specifically, however, is a topic of debate. ADELFE, on the other hand, puts the development of algorithmic, low-level solutions at the forefront of the development process.

An important disadvantage of many approaches that is cited in a number of comparison papers is a lack of specialised tools for the process (see, e.g., Cossentino et al., 2011). We do not consider this a disadvantage as long as the modelling languages used have a strong tool support. If the requirements are modelled in KAOS and the design documents are modelled in UML, respective modelling tools can be used and a specialised tool set is not necessary. On the contrary, a specialised tool can mean a strong dependence on the further development of the tool and that the developers are shut out of advances in standard software. As an example, the MaSE methodology can only be properly applied when "agentTool"—a graphical development environment—is used since only this software supports the full modelling environment. However, this means that a developer will have to use the tools provided with this software and can not benefit from advances made, e.g., in the context of model transformation languages, or use a different modelling language. Especially the use of model transformation techniques and the application of a model-driven design approach can be hindered by non-standard tools.

If standards are used instead, it becomes possible to use a wide variety of modelling tools and the infrastructure that accompanies them. If a standard modelling language like UML is used, e.g., specialised UML profiles and the use of OCL constraints with appropriate checkers can simplify semantic consistency checks. Model transformations and code generation can be defined in a variety of languages. The integration of domain-specific languages designed by the development team and the connection to specialised tools, e.g., model checkers or simulation environments, can be achieved by using standard interfaces. The use of such accessories is, of course, at the discretion of the development team.

Chapter Summary and Outlook

This chapter introduced the methodological foundations of method and process engineering that are the basis for any attempt to create a new software engineering methodology. It also illuminates the current trend in agent-oriented software engineering, the sub-discipline that deals exclusively with the special circumstances present when designing multi-agent systems and outlines existing AOSE processes that define the state of the art in this area. Based on requirements for large-scale open self-organising systems, these existing processes are evaluated on their suitability for the system class and their disadvantages are identified. These findings are the foundation of a specialised methodology and the accompanying method content introduced in Chapter 13 and evaluated with two case studies in Chapter 14.

Chapter 13

PosoMAS: A Custom AOSE Methodology to Accommodate Openness and Self-Organisation

Summary. This chapter introduces PosoMAS, the **P**rocess for **o**pen, **s**elf-**o**rganising **M**ulti-**A**gent **S**ystems. The process is composed of a number of practices, reusable and customisable building parts, and integrated into the framework of the Open Unified Process to yield an iterative, incremental software engineering process tailored to the specific class regarded in this thesis. The individual practices are briefly introduced and their interplay is described. Possibilities for the customisation of the process for specific project settings are discussed as well. In addition, insights into embedding the practices into other processes is exemplified by combining the PosoMAS practices with Scrum.

Publication. Relevant preliminary work on the subject has been published in (Sudeikat et al., 2010), (Seebach et al., 2010), and (Sudeikat et al., 2012).

The **P**rocess for **o**pen, **s**elf-**o**rganising **M**ulti-**A**gent **S**ystems (PosoMAS) aims to fulfil the requirements outlined in the previous chapter with an emphasis on open self-organising multi-agent systems and a focus on customisability and applicability to a number of method life-cycles. It strives to be open and readily available in a standardised format and is therefore defined through a number of practices, modelled in a standard format that makes them easily accessible to method engineers and allows their combination with other practices as described in Section 13.1. The practices contained in the PosoMAS practices library operationalise the techniques developed in this dissertation and add specific roles, guidelines, activities, tasks, and categories for the target system class that take into account the requirements outlined in Chapter 12 as well as those introduced in Chapter 2 such as modularisation, compartmentalisation and emphasis on interfaces. The practices are briefly introduced in Section 13.2. However, to create a consistent software engineering process, these practices are not sufficient. They need to be combined with a project life-cycle and further practices that define the management tasks and the basic structure of the process. Section 13.3 thus introduces the Open Unified Process (OpenUP) and the Eclipse Process Framework (EPF) practices library that provide method content for iterative-incremental development and a risk-value life-cycle. The practices from the PosoMAS practices library and from the EPF practices library are then combined to create a process that is structured similarly to the OpenUP and addresses the specific requirements of open self-organising MAS in Section 13.4. The resulting process is checked against these requirements in Section 13.5. This approach brings AOSE closer to what is possible in classical, object-oriented software engineering, where designers are provided a framework in which they can define their own path (Xiao and XueYan, 2008). Finally, possible customisations of the process and combinations with the light-weight agile methodology Scrum are discussed in Section 13.6.

13.1 Elements of the AOSE Methodology

PosoMAS is the product of an assembly-based process model (Ralyté et al., 2003). It uses practices as its foundational building blocks that define roles, tasks, activities, work products and other relevant elements of the methodology. Each practice focuses on a specific aspect of open self-organising multi-

agent systems and provides a self-contained operationalisation of concepts and techniques relevant for this class of systems. The practices used in the assembly of the process are a result of the analysis of existing processes and the operationalisation of techniques developed within the scope of this thesis and the research project it is embedded in.

Practices as the Building Blocks of PosoMAS

The distinction between the library that contains the practices and the actual processes is important. The current guidelines for method authoring provided by the Eclipse Process Framework community (available at `http://epf.eclipse.org/wikis/mam/`) propose the definition of all method content within practices that are in turn contained in practice libraries. The content from one or more of these libraries is then used to define a process. This guideline has been followed with the creation of the PosoMAS Practices Library introduced in Section 13.2. It is combined with the EPF Practices Library containing the OpenUP life-cycle, management activities, and method content concerned with testing and deployment. This content has been used by the Eclipse community to define the OpenUP which is described briefly in Section 13.3. Combining the PosoMAS practices with the EPF practices yields the process contributed by this thesis—PosoMAS—whose structure is detailed in Section 13.4. Similarly, the combination of the PosoMAS practices with other process frameworks yields different processes, e.g., agile or heavy-weight ones. Combining the PosoMAS practices and Scrum as an example of a light-weight agile process is discussed in Section 13.6.

Practices are designed in SPEM[1] and modelled with the Eclipse Process Framework Composer. The main concepts of SPEM are defined in the box on page 191. Using this accepted standard and a wide-spread and openly available tool for modelling allows the reuse of the practices and the method content they contain, thus addressing the requirement for extensibility and customisability and enabling the reuse of the developed method content and its publication in repositories.

Process life-cycle

The process life-cycle determines the progression of a product under development in the context of the process, the stakeholders, and the environment. A well-defined life-cycle provides guidance w.r.t. the order of activities the development team has to perform, the project milestones and deliverables, and the progress of the product. The advancement of a product development effort can thus be measured based on the planned and the actual progress within the life-cycle.

A process is created by embedding the activities and tasks defined in the practices into a life-cycle. The structure the life-cycle provides is often defined as phases (e.g., inception, elaboration, construction, and transition in the OpenUP as described in Section 13.3) that are executed sequentially. Each phase addresses different needs within the project and in general, a shift away from requirements elicitation, towards design, and then implementation and testing is evident. Phases can themselves be subdivided into smaller elements.

The PosoMAS practices are embedded in the risk-value life-cycle of the OpenUP (cf. Section 13.3). It promotes an approach in which the most risky requirements and those that provide the greatest value are tackled first by the development team in an iterative-incremental way, in which each phase consists of a number of iterations. Section 13.6 shows an example of how the practices can be embedded in a process that only provides a rather unstructured pre-defined life-cycle.

Scope of the Process

While PosoMAS has been designed to support the development of large-scale open self-organising systems and to fulfil the requirements outlined in Section 12.3, there are some requirements met by other AOSE methodologies that the process outlined here does not address. Some of these requirements contradict the design principles of PosoMAS, e.g., independence of a certain agent architecture. Thus, instead of regarding these "missing" elements as limitations of PosoMAS, they are simply outside the scope of the process. Among the activities and requirements PosoMAS does not address are:

[1] Software & Systems Process Engineering Metamodel (`http://www.omg.org/spec/SPEM/2.0/`), defined by the Object Management Group and widely adopted for the definition of software engineering processes. The Eclipse Process Framework (EPF) (`http://eclipse.org/epf`) is a powerful tool to create SPEM models and processes based on practices defined this way.

- Regard percepts and actions specifically as done in Prometheus (Padgham and Winikoff, 2005). Data coming from noisy sensors has to be handled explicitly (as, e.g., in the ERS scenario), but the process does not offer specific tools for this purpose.
- Provide guidance for the design of BDI agents, i.e., guidance for the development of mental attitudes and beliefs. This can be imported from other processes, however.
- Incorporate specialised meta-models for system classes or agent architectures. As mentioned before, meta-models can be helpful when developing systems for specific classes by providing guidance about the necessary design elements and their relationship. PosoMAS has been designed to be independent of a concrete meta-model, thus ensuring its applicability for a wide range of systems.

If such features are required, however, they can be imported from other processes that provide appropriate method chunks and combined with the PosoMAS practices during the process customisation step. This includes tailoring the practices of the PosoMAS to accommodate additional content or roles. Specialised meta-models can easily be integrated into the design steps.

The Key Concepts of SPEM

SPEM includes a number of concepts that are defined in the method content. They are abstract definitions (similar to the classes in a UML class diagram) of tasks, work products, roles, and so on. This method content is then used in the definition of a process—i.e., a sequence of steps to be taken in the development of a product—thus defining task uses, work product uses, etc., as depicted in Figure 13.1. These uses are similar to instances in standard object-oriented UML but are defined as proxies for the appropriate definitions. Thus, a use has an association to the appropriate definition. Uses can be altered in comparison to the original definition, however, e.g., by selecting a subset of the steps defined in a task definition in a concrete task use. In the following, the key concepts of SPEM are introduced. For a more detailed description, especially how the concepts relate to the SPEM meta-model, please refer to (OMG, 2008). The description provided here is based on this specification unless otherwise noted.

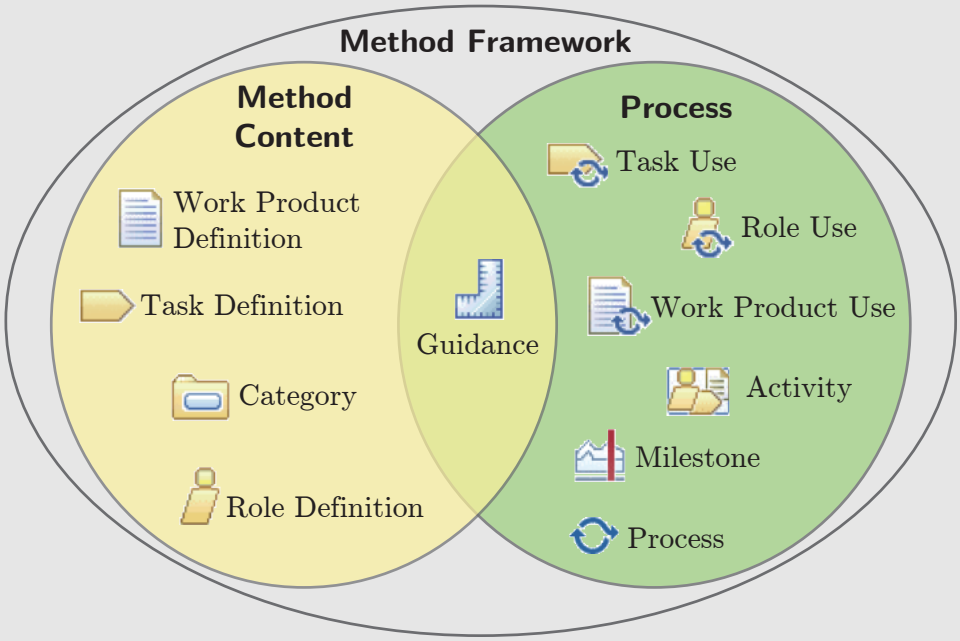


Figure 13.1: SPEM's key concepts and their assignment to method content and process definition. The concepts used in the process definition are instantiations of the concepts that are part of the method content. Picture based on (OMG, 2008).

Concept	Assignment	Description	Icon
Task	Method Content	Describes a unit of work. A task can create or transform a work product and can be assigned to a certain role. It can contain individual steps that have to be performed as part of carrying out the task. Guidances can be provided, e.g., to give guidelines or checklists. Tasks can be subsumed in activities.	
Task Use	Process	Instantiates a task in the context of an activity. It is possible to define a subset of the task's steps for execution at this stage within the activity.	
Activity	Process	Groups other elements such as method content uses, milestones, or other activities. Thus, an activity is both a building block of a process or a process in itself. The elements within an activity can be linked to role uses, work product uses, and process parameters. The order of elements within the activity is denoted as the *work breakdown structure* and is established by defining predecessors for the elements in the structure. The "Use"-elements below are always defined within the context of an activity.	
Activity Use	Process	An activity embedded as part of the work breakdown structure of another activity. This kind of reuse allows the construction of complex activities and so called *capability patterns* that capture a structured part of a process.	
Milestone	Process	Defines a significant event in the process. Milestones often occur at the end of iterations or at the end of phases, e.g., as prototypes or deliverable documentation. They can be embedded in activities.	
Work Product	Method Content	Documents, models, code, or other tangible elements that are produced, consumed, and modified by tasks. Responsibilities for work products can be assigned to roles.	
Work Product Use	Process	Defines the occurrence of a work product within an activity as its input or output. Can redefine the original relation of the work product to tasks and roles.	
Work Product Slot	—	Although not a part of SPEM, work product slots play an important role in the definition of work products that are exchanged between practices. As an example, the requirements can be captured in a number of concrete work products, e.g., in a use-case model, in user stories, or in a systems goal model. Using the more generic work product slot *[Requirements Model]* allows specifying that requirements are used by a practice but does not prescribe which kind of requirements are necessary. Work product slots are fulfilled with concrete work products in the assembly of the process.	
Role	Method Content	Denotes an individual or a group of individuals with a certain set of skills, competencies, and responsibilities required in the process. Different roles can be filled by different people during the process and an individual can fill several roles if required.	
Role Use	Process	Represents the performer of a task within an activity. A role use specifies that a certain task has to be performed by one of the team members, possibly someone holding the appropriate qualification. In agile environments, where team members decide autonomously which roles to take on, it might not be necessary to back a role use with a formal role definition.	

Process	Process	Defines the structure of activities and tasks that determine the order in which sequences of work are performed and phases and milestones are completed to get to the final product. Within a process, concrete role uses, task uses, work products, etc. are defined. A customised process for a specific project is modelled as a *Delivery Process*.	
Guidance	Managed Content	Provides additional information about the elements in the method content. Different kinds of guidances are possible: guidelines, templates, checklists, tool mentors, supporting materials, reports, concepts, examples, and others.	
Category	Managed Content	Organises content according to domains, topics, disciplines, phases, or other criteria.	

Table 13.1: Important SPEM concepts, their assignment to one of the SPEM packages, their description, and graphical representation.

The method content and its use is defined in *method plugins*. They allow bundling related content and its reuse in different processes. Plugins can in turn be bundled in *method configurations* that combine related method elements. *Method libraries* can contain both these elements.

SPEM borrows some UML diagram types to graphically define certain aspects of the method content and the process. UML activity diagrams are used to describe the sequence of tasks performed in activities. The evolution of work products can be defined with UML state machines and the corresponding diagrams if appropriate—e.g., if a work product can change its status from "preliminary" to "draft" to "final".

13.2 Practices for Large-Scale Open Self-Organising Systems at a Glance

The practices for PosoMAS cover the disciplines requirements and analysis & design, with an emphasis on the latter. Testing and deployment are indeed the focus of ongoing work since both disciplines are of utmost importance in multi-agent systems and have not been dealt with sufficiently as of yet. The practices introduce techniques for the principled design of agents, systems, their interactions, and the environment. These factors are captured in different practices within the architecture discipline and are an important factor in the overall design of the process. Four areas are distinguished:

Agent Architecture: the design of the individual agents, separate for each type of agent.

Agent Organisation: the specification of an organisational paradigm that will structure the agent population at runtime.

Agent Interaction: the definition of interfaces and protocols used by agents to exchange information, delegate control, and change their organisational structure.

System Architecture: the relationship between the different types of agents, the supporting infrastructure, and external components as well as the environment.

These areas are connected to each other and it is necessary to define common work products that can be used to exchange information between the activities and tasks specified for each of the areas. To structure these work products, respective SPEM domains and disciplines have been introduced. The "Agent Architecture" and "System Architecture" areas are combined in an *architecture* domain and discipline. Agent interactions and agent organisations are captured in a respective domain and discipline as well.

Common Categories, Work Products, Roles, and Domains

The PosoMAS practices library introduces a number of work product slots, an additional role, and specialised domains. This method content allows the categorisation of artefacts and work within the development effort and provides a structuration of work products and tasks.

Work Products Common work products are defined as *work product slots*, placeholders for actual work products. They are used in the exchange of information between practices and capture the different architecture areas defined by PosoMAS. Using slots allows a method engineer to replace a slot with a concrete work product. Requirements can, e.g., be captured with a system goal document or with a use case model. Instead of defining which concrete work product to use, a work product slot only defines that a *[Requirements Model]* has to exist. In the customisation of the process, the method engineer can then use whichever concrete work product is relevant in the process. The square braces around the name of the work product slot indicate by convention that a slot is indeed used. To exchange information about the agent architecture, a work product slot *[Agent Architecture]* is used. It serves as an abstraction of high-level artefacts that represent the documentation of the architecture of a single agent within a multi-agent system. The architecture of the system itself is documented by the work product slot *[Multi-Agent System Architecture]*. The environment in which the agents operate is captured in the *[System Environment Description]*. The *[Agent Organisation Model]* captures the system structure that the agents are arranged in to fulfil the requirements. Finally, the *[Interaction Model]* serves as an abstraction of models that describe the interactions between agents, particularly the protocols involved and the interaction media. The tasks defined in the practices use these slots as their inputs and outputs. Additionally, some practices define work products that are only used to exchange information between tasks within the practice. In such cases, slots are not used.

Role: Product Owner. The product owner is a special kind of stakeholder and represents the client in the development process. The concept has been borrowed from Scrum, where the product owner is embedded into the development effort at all times and is on the one hand responsible for prioritising work and on the other hand serves as the domain expert the development team can direct questions to. In the context of the PosoMAS, the product owner is mainly involved in the requirements elicitation process and in the aspects that relate to the environment the system will work in. Please note that the EPF practices library also defines a *Product Owner* role. This definition and its accompanying description, however, is very much focussed on Scrum and explicitly contains elements that are not part of the PosoMAS or the OpenUP.

Other Roles. The OpenUP defines a number of roles that fulfil certain tasks in the process and require a certain skill set. A *Project Manager* is responsible for planning the project as a whole as well as each iteration and control the progress. He is supported by a *Process Engineer* in the tailoring of the process. The *Analyst* is the liaison between the development team and the customer and communicates with the latter to capture requirements and discuss issues. The responsibility for the software architecture and for most non-trivial design tasks is with the *Architect* who also oversees important implementation decisions. The implementation itself is performed by the *Developer*, including testing, UI design, and build integration. In addition to these core roles, there are a number of supportive roles, such as *Tester*, *Deployment Engineer*, or *Technical Writer* that can play a role at some stages of the process. In general, a small development team will not have specialists for each of the roles, but each member of the development team will at one point fulfil different roles within the project.

Domains. PosoMAS introduces four domains, specialised categories for the classification of work products that relate to the different areas of the development effort. Work products can be related to *Agent Interaction* and *System Organisation*. In addition, the *Requirements* and *Architecture* domains from the EPF practices library are supplemented with domains that contribute content to them.

Overview of PosoMAS practices

As PosoMAS is targeted at open systems, the architectural tasks are aimed at providing standardisation, separation of concerns, and interoperability. The applicability to a wide range of target systems has also been a concern. Therefore, even though the practices are specific to open self-organising multi-agent systems, they do not require the use of a specific meta-model or agent architecture. The practice library provides the following practices, each of which is described below in more detail with the respective tasks, work products that are used as inputs and outputs, and the guidelines that are provided:

Goal-driven Requirements Elicitation: operationalises the technique for requirements elicitation based on KAOS (Lamsweerde and Letier, 2004) and the work of Cheng et al. (2009) introduced in Section 7.2. The practice is described in more detail below.

Tasks: Identify System Goals, Refine System Goals to Requirements, Mitigate Uncertainty Factors, Define System Limitations and Constraints, Validate Requirements
Input: Vision (optional), [Requirements Model] (optional)
Output: System Goals Document, Conceptual Domain Model, Glossary, [Requirements Model]
Guidance: Goal-Driven Requirements Engineering with KAOS (Supporting Material), System Scope and Requirements (Supporting Material), The Scope of Requirement Elicitation (Guideline), Objectiver (Tool Mentor)

Model-driven Observer Synthesis: describes how observer implementations can be synthesized from constraints specified in the requirements documents as described in Chapter 6 and Chapter 7. The details of this practice are elaborated below.
Tasks: Define Observation Model, Create Transformation to Observation Model, Create Transformation to Observer Implementation, Perform Transformations
Input: [Agent Architecture], [Interaction Model], [Multi-Agent System Architecture], [Requirements Model]
Output: Observation Model, Transformation rules to transform constraints into an observation model, Transformation rules to transform observation models to observer implementations, Implementation
Guidance: How to adopt the Model-driven Observer Synthesis practice (Roadmap), Observation Model (Concept), Observer/Controller Architecture (Concept), Synthesis of Observers for Autonomic Evolutionary Systems from Requirements Models (Whitepaper)

Pattern-driven MAS Design: provides guidance to design a multi-agent system based on existing architectural and behavioural patterns such as the ones introduced in Chapter 3. A design conscientious of existing work enables reuse, avoids making mistakes twice, and allows tapping into the knowledge that has been created elsewhere for a cleaner, leaner, and more flexible design. An architectural pattern can be applied in the development of the system architecture, while more fine-grained patterns and protocols can be used to create agent architectures and define interactions between agents. The use of patterns also facilitates communication between stakeholders and makes the architecture and the implementation more comprehensible for developers and architects.
Tasks: Apply Architectural Style, Apply Patterns to Agent Architecture, Apply Patterns to Agent Interactions
Input: [Agent Architecture], [Interaction Model], [Multi-Agent System Architecture], [Requirements Model]
Output: [Agent Architecture], [Interaction Model], [Multi-Agent System Architecture], Architectural Style Notebook
Guidance: How to adopt the Pattern-Driven Design practice (Roadmap), A Catalogue of Architectural Styles (Supporting Material), Literature on Patterns for Agent Architectures (Supporting Material), FIPA Interaction Protocols Specification (Supporting Material), Architectural Pattern (Concept), Reference Architecture (Concept), Anders et al. – Reference Architectures for Trustworthy Energy Management, Desktop Grid Computing Applications, and Ubiquitous Display Environments (White Paper), Paris Avgeriou & Uwe Zdun – Architectural Patterns Revisited: A Pattern Language (White Paper)

Evolutionary Agent Design: describes an approach to design agents and their architecture in an evolutionary process that will enhance the agents over time while the requirements become more clear and the development progresses. During the development process, the agent types, their capabilities and behaviour, their internal architecture, and their interactions will become successively clearer as the requirements mature and the system design progresses towards a shippable build. To allow the product to mature this way, the design of the agents has to adapt to new knowledge continuously and become more specific by refining the design when necessary and incorporating changes in the requirements or the system environment.

Tasks:	Define Agent Capabilities and Behaviour, Design Agent Architecture, Design Agent Interactions
Input:	[Agent Architecture], [Interaction Model], [Multi-Agent System Architecture], [Requirements Model]
Output:	[Agent Architecture], [Interaction Model], [Multi-Agent System Architecture]
Guidance:	Modular Agent Design (Guideline), Agent Capability (Concept), PosoMAS Auxiliary UML Profile (Reusable Asset)

Agent System Design: outlines how the system the agents are embedded in is designed and how the agents interact with it. A multi-agent system not only consists of autonomous agents but also incorporates a multitude of additional infrastructure, external actors, interfaces, and environmental factors. These can have a significant impact on the overall system design and should be regarded in the overall system design early on.

Tasks:	Identify Required Supporting Infrastructure, Identify System Environment, Model External Interfaces, Define System Architecture
Input:	[Multi-Agent System Architecture], [System Environment Description], [Interaction Model], [Requirements Model]
Output:	[Multi-Agent System Architecture], [System Environment Description], [Interaction Model]
Guidance:	Separation of System and Agent Levels (Guideline)

Agent Organisation Design: describes the design of the organisation the agents will interact in at runtime, based on the organisations introduced in Chapter 3 and Chapter 4. Multi-agent systems with many interacting agents require a form of structure or organisation imposed on the population. Sometimes, this structure is permanent, such as a hierarchy that determines the delegation of control and propagation of information, or transient, such as a coalition in which agents interact until a certain common goal is reached. The system designer has to decide which organisations are suitable for the system to reach the stated goals and implement mechanisms that allow the formation of these organisational structures at runtime. If this process is driven from within the system, self-organisation is present.

Tasks:	Identify a Suitable System Organisation, Specify Self-Organisation Algorithm
Input:	[Multi-Agent System Architecture], [Requirements Model]
Output:	[Agent Organisation Definition]
Guidance:	Bryan Horling and Victor Lesser – A Survey of Multi-Agent Organizational Paradigms (Whitepaper)

Trust-based Interaction Design: guides the design of interactions in multi-agent systems that can be evaluated with trust models and agent decisions that use trust values to make the system more robust and efficient. Trust-based interaction design enables the agents in the system to determine and select trustworthy interaction partners with a high likelihood of successful completion of an interaction. It helps make the system more robust against unintentional and malevolent interaction behaviour and can even enable more efficient problem solving.

Tasks:	Identify Sources of Uncertainty, Define Trust Model, Define Agent Decisions based on Trust Values, Design Trust Measurement Infrastructure, Design a Reputation System
Input:	[Agent Architecture], [Multi-Agent System Architecture], [Requirements Model], [System Environment Description], [Interaction Model]
Output:	Trust Model, [Multi-Agent System Architecture], [Agent Architecture]
Guidance:	How to adopt the Trust-based Interaction Design practice (Roadmap), Trust in Multi-Agent Systems (Concept), Reputation in Multi-Agent Systems (Concept), Trust in Organic Computing Systems (Concept), Bibliography for Trust in Multi-Agent Systems (Supporting Material), The Trust Lifecycle (Supporting Material)

Each practice is defined by an appropriate guidance in EPF that states the purpose of the practice, gives a description, and provides a brief instruction on how the elements of the practice relate to each other and in which order they should be read. The practice usually references a roadmap (another special type of guidance) for the adoption of the practice, a list of key concepts and white papers, several work products that are altered or created by the tasks and a set of guidances. A practice also takes one or several work products (or work product slots) as inputs. These inputs are automatically derived from the respective relationships of the tasks. Work products that are denoted as inputs are at

the same time often outputs of the tasks. If the practices are combined into a process, the outputs of the practices can be used to instantiate the work product slots denoting the inputs of the other practices. The *Conceptual Domain Model* can, e.g., be used to fill the [Multi-Agent System Architecure] in early iterations of the process.

The detailed description of the practices is available online at `http://practices.oc-trust.isse.de`. The models for use in EPF are also available at this URL. We thereby provide a repository for method content and make reusable assets available for combination with method content from other processes, thus fulfilling the appeal of the IEEE FIPA Design Process Documentation and Fragmentation Working Group and many authors (see, e.g., (Seidita et al., 2006; Cossentino et al., 2011), and Section 12.1). In the following, the first two of the practices listed above are summarised and their most important elements explained. The other practices, the tasks and guidances they contain, and the way the are applied in the process will be discussed in the validation of the process in Chapter 14.

Practice: Goal-driven Requirements Elicitation

When defining the requirements of an open self-organising system, the goals of the system are paramount. The process of refining them to requirements for the individual agent types will reveal necessary interactions, the obstacles the system has to deal with, and additional system goals that need to be in place to make the system adaptive. A clear picture of the system's goals also defines the scope of the system. Only requirements that contribute directly to attaining the system goals should be defined and elaborated during design and implementation. Therefore, this practice can also facilitate the communication between Product Owner and the Development Team: only system goals defined consensually and refined accordingly are going to be implemented. It is also possible to explore sub-goals and mark them for implementation during the next iteration. Such conventions clearly define the targets for the next iteration and make it easy to manage expectations.

Concept: System Goal System goals are usually defined by the Product Owner and other stakeholders and describe the objectives the system has to attain within the environment it is going to be deployed in. Such goals can be refined into more fine-grained sub-goals that describe the different aspects that have to be taken into consideration to achieve the over-arching goal.

Top-level system goals are defined early on in the process and specify the scope of the system. They are often the basis for the contract between the client and the development company and should thus be relatively stable. Changes in the top-level system goals usually propagate down the goal hierarchy and can therefore have disruptive consequences.

Work Product: System Goals Document Captures and describes high-level system goals, their refinement to lower level goals, the requirements that result from these goals, the agent types the requirements are assigned to, and the obstacles that can prevent the goals to be reached. The system goals document captures all relevant aspects of the functional and non-functional requirements of the system. It is used during the entire process to guide design and implementation and communicate with the customer. It can be used in other practices to fill the [Requirements Model] work product slot.

Work Product: Conceptual Domain Model Captures the most important concepts in the domain the system is developed in as well as their relations. The Conceptual Domain Model is a structured specification of the relevant concepts of the system domain. It captures the relevant concepts and their relationship to each other, as well as some of the attributes and operations that have to be performed in the system. It is often executed as a UML class diagram and can be an input to the tasks that define the architecture of the system.

Task: Identify System Goals Define the purpose of the system and the over-arching functionality it has to fulfil. Performed by the Analyst and the Product Owner, with optional support from the Project Manager and other Stakeholders. Creates the System Goals Document. Part of the Requirements discipline.

Steps: The main steps of this task consist of *gathering information* from the stakeholders, specifically the product owner that can be used to *identify top-level system goals* that are recorded in the corresponding documents. These top-level system goals are *refined to more granular goals.* Next, *obstacles that can hinder the system from reaching its goals are identified.* These can be environmental,

regulatory, or technical and might prevent certain goals from being reached. Finally, *assumptions about the environment and the system are identified* and recorded in the System Goals Document.

Task: Refine System Goals to Requirements Uses system goals as the basis for a refinement towards requirements for the different agent types. Performed by the Product Owner, the Project Manager, and the other Stakeholders. Uses the Requirements Model and optionally the System Goals Document. Creates the Requirements Model, the Conceptual Domain Model, and the Glossary. Part of the Requirements discipline.

In an ideal case, all stakeholders work together to find all the requirements the system has to satisfy. As a crucial part of the refinement process, the high-level system goals have to be broken down to sub-goals and finally to requirements. This should be a joint effort of the Product Owner, the Stakeholders, and the development team. During this process step, the final functionality of the system is defined and the correct and concise formulation of goals and requirements is established. At the same time non-functional goals and requirements can be identified, e.g., for the performance of the system, its usability, or its reliability.

Steps: First, the high-level goals to work on are selected. These high-level goals are then refined to sub-goals and requirements. A requirement is a goal for which a certain type of agent is responsible. Thus, agent types are identified while the goals are refined. The goals and requirements give indications for certain system elements that are important. These concepts are captured in the conceptual domain model. Important terms are described in the glossary. Finally, additional obstacles are identified and recorded.

Task: Mitigate Uncertainty Factors Adapt the system goals and requirements to deal with obstacles and factors of uncertainty. Performed by the Analyst and the Developer, with optional support from the Product Owner. Uses and alters the Requirements Model and the Conceptual Domain Model. Part of the Requirements discipline.

There are several sources of uncertainties that can be distinguished in open multi-agent systems. First of all, the behaviour of the system user is uncertain, especially if there are complex and numerous ways to interact with the system. A user could, e.g., alter the environment and thus indirectly influence the system. Second of all, external components can behave unreliably or unexpectedly. Third of all, data that is gathered about processes in the environment or within the system can be affected with uncertainty. Cheng et al. (2009) suggest four different mitigation strategies for uncertainties, introduced in Section 7.2. These strategies can be applied within this task.

Steps: Identify uncertainty factors by starting from the requirements and using the previously defined obstacles and uncertainties. Not all of these obstacles are relevant for the system so it is necessary to identify which uncertainty factors the system has to react to. If data is afflicted by uncertainties, denote the data in the conceptual domain model. Finally, uncertainty factors are mitigated with one of the possibilities outlined in Section 7.2.

Task: Define System Limitations and Constraints Based on the expressed wishes of the stakeholders, the developer is now able to define the limitations and constraints that these requirements mean for the system to be built. Performed by the Developer with optional support from the Stake Holder. Uses the Requirements Model and optionally the Conceptual Domain Model and alters the former. Part of the Requirements discipline. Optional task. Recommended if the practice Model-driven Observer Synthesis is used or formal verification has to be performed.

An adaptive, open system has to be able to observe itself and its environment to identify system states in which undesirable behaviour can occur or in which the system functionality can not be upheld. Such conditions can be identified during requirements analysis and (semi-)formally described for use later on in the process. The formal description uses concepts, associations, and fields of the agents as captured in a domain model. During the process of formalising the constraints, additional concepts can be identified and added to this artefact. The formal definition of the system's limitations can also facilitate the identification of so far hidden assumptions about the agents or the environment. If such an assumption is identified, the stakeholders can discuss it during the next iteration and either adapt the requirements accordingly or ensure during deployment that the assumption holds. This task resembles the operationalisations of goals into constraints as described in (Dardenne et al., 1993).

Steps: First, the requirements that can be expressed as constraints are identified. These system constraints are formalised and the respective requirements are annotated. In the course of the formal-

isation it might become necessary to extend the domain model and the requirements with additional elements that have not been captured so far.

Task: Validate Requirements As a final task, the validation of the requirements ensures that they are unambiguous, not contradictory and clearly phrased. Performed by the Analyst, Architect, Developer, and Product Owner. Requires the Requirements Model and optionally the Conceptual Domain Model and the System Goals Document. Part of the Requirements discipline.

Conduct a review of the requirements with relevant stakeholders and the development team to ensure consistency with the agreed vision, assess quality, and identify any required changes. At this point, make sure that the identified requirements are refinements of the system goals and that the requirements are able to fulfil the system goals.

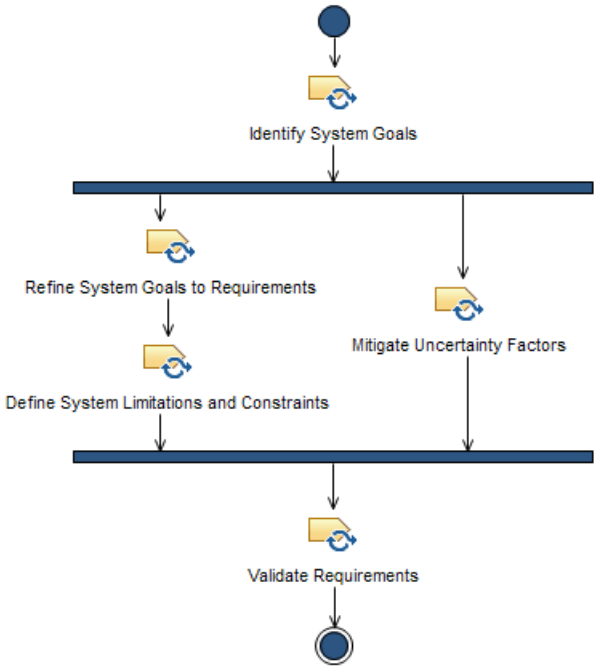


Figure 13.2: The activity diagram for the practice Goal-driven Requirements Elicitation. The first step is to define top-level system goals that are then successively refined to requirements. These requirements can be annotated with formal constraints. Concurrently, obstacles and uncertainties can be identified and corresponding measures to mitigate them can be introduced into the requirements model. Finally, the requirements are validated with the stakeholders. The activity can be repeated and the requirements are thus successively refined. Apart from the requirements model, a system goal model capturing the high-level goals, a conceptual domain model capturing important concepts that come up, and a glossary explaining important terms are created.

Practice: Model-driven Observer Synthesis

In adaptive systems, it is necessary to observe the system and react if the system enters an undesirable state or shows unwanted behaviour (cf. Chapter 6). For this purpose, feedback loops, operationalised as observer/controllers can be employed (cf. Chapter 8). This practice describes an automatic transformation to observer/controller implementations from constraints defined during requirements analysis (cf. Chapter 7). A prerequisite of this practice is that constraints have been captured during requirements analysis. Ideally, these are expressed as OCL constraints that define the correct states of the system. If the Define System Limitations and Constraints task from the practice Goal-driven Requirements Elicitation is performed, constraints should be available. At the same time, this ensures that a domain model containing the elements the constraints are defined on is available. Constraints can also be defined in a specialised document separate from the requirements model. The process can be repeated after the requirements or the domain model have changed, according to a model-driven design (MDD) approach. Changed parts of the system models and implementation will be re-generated while existing models and code are preserved.

Work Product: Observation Model The observation model describes the monitoring infrastructure, its relationship with the agent and system architecture and the interaction between these elements.

In order to be able to monitor the system and the constraints that it has to adhere to, a specialised infrastructure has to be defined. This infrastructure is captured in the observation model. It usually consists of monitors or observers that acquire data about the state of a specific agent, a group of agents, or parts of the system and check whether the specified constraints hold. Additionally, it contains

controllers that react to the violation of these constraints in order to restore the constraint in question. This distinction is detailed in Observer/Controller Architecture and can be found in other kinds of operationalised feedback loops such as the MAPE cycle.

The observation model can be related to the agent architecture—in case the constraints can be observed on the agent level—or to the system architecture in case the constraints need more global data to be observed. This corresponds to the different varieties of the Observer/Controller Architecture, in which the observer and the controller respectively can be located at different levels of the system. In many cases, several observer/controllers on different levels of the system are required to handle different constraints.

The observation model contains a number of vital design choices. If all agents are implemented by the development team, it might be possible to integrate the monitoring infrastructure with the agents. If, however, a blackbox view of the agents is necessary, e.g., because the system is open and agents provided by many vendors interact with each other, it might only be possible to observe the agents externally, i.e., with data provided by them voluntarily. In such a case, the formation of models of the agents is necessary, as well as a process to update them continuously.

Task: Define Observation Model The model of the observer, the controller, and the relation to the agents has to be adapted to the system. Performed by the Architect, with optional support from the Developer. Creates the Observation Model. Part of the Development discipline.

Steps: First, identify and define relevant classes, including observers and controllers as well as agent states. Next, the interaction between observers and agents has to be defined, specifying how the observer gathers state information from the agent. Likewise, the interaction between observer and controller are defined, specifying under which conditions and with which interface the observer triggers the controller. These considerations become part of the architecture by defining relevant associations and fields of the classes in the observation model. Finally, the behaviour of the relevant methods is specified with appropriate models.

Task: Create Transformation to Observation Model Define a model-to-model transformation, creating classes and model elements that extend the observation model based on the constraint specification. Performed by the Developer. Uses the Observation Model and optionally the Agent Architecture, the Multi-Agent System Architecture, and the Requirements Model. Creates the Transformation rules to transform constraints into an observation model. Part of the Development discipline.

Steps: First, create transformation rules for specialisations of observers and controllers. These rules have to check whether an agent has constraints associated to it and create specialised observers and controllers for these constraints. Then, rules for the constraints themselves have to be created. These rules will create classes that embed the constraints and can be checked by the observers on the current agent state. Finally, create rules for the specialisation of the dynamic behaviour, e.g., the check whether a constraint holds.

Task: Create Transformation to Observer Implementation Create a model-to-text transformation that transforms the observation model to an observer/controller implementation for the target system. Performed by the Developer. Uses the Observation Model and optionally the Agent Architecture, the Multi-Agent System Architecture, and the Requirements Model. Transformation rules to transform observation models to observer implementations. Part of the Development discipline.

This task defines a transformation from the previously extended observation model to the actual source code that implements the monitoring infrastructure. Since the implementation is highly specific for the specific system and target implementation platform (e.g., the used multi-agent system or middleware), these rules have to be tailored for the target platform and system. However, some of the basic principles remain the same regardless of the transformation target.

The classes and sequence diagrams have to be translated into the target programming language and the target platform, i.e., the multi-agent platform or middleware the system will run on. Sequence diagrams become implementations of the methods of the classes. The flow of information from the agents to the observers and from there to the controllers has to be captured as defined in the observation model and adapted to the communication infrastructure provided. A decision has to be made, whether or not observers and controllers are independent agents or become part of the agents defined in the domain model, whether properties of the agents can be accessed directly or only via message passing, etc.

Steps: First, implement the parts of the observation model that do not change depending on the agents and constraints at play and therefore do not have to be created by a transformation. Then, create transformation rules for class stubs and finally create transformation rules for attributes and methods.

Task: Perform Transformations Perform the transformation from constraints to the observer implementation. Performed by the Developer. Uses the Observation Model, the Transformation rules to transform constraints into an observation model, and the Transformation rules to transform observation models to observer implementations and creates part of the Implementation. Part of the Development discipline.

In the final step, the transformations are performed to first extend the observation model and then transform this extended model into source code for the monitoring infrastructure. In addition, the initialisation of the monitoring infrastructure has to be considered.

13.3 The Open Unified Process as a Method Template

The basis for PosoMAS is the *Open Unified Process* OpenUP (Eclipse Foundation, 2013; Gustafsson, 2008). Its original content was based on a donation of content from the commercial Rational Unified Process (RUP) (Kruchten, 2004), but it has since been developed by the community towards a lean, agile, process and—more importantly—towards an extensible process framework that provides a minimal starting point for customisations and extensions. The original RUP is a heavy-weight process that provides detailed instructions for software development, based on a vast collection of best practices. Its strength is the vast knowledge it provides for almost all conceivable situations that can occur during the development of a software product. This, however, turns out to be a weakness as well: as pointed out by Gustafsson, RUP is cumbersome to work with as it is difficult to find the information that is needed at any particular moment. Due to its size and complexity, it also has a steep learning curve that makes it difficult to adopt. At the same time, agile processes have received quite a lot of attention. Their focus on early delivery, customer involvement, and minimal documentation have made them popular with software developers and clients. However, such processes are often unpopular with management since they provide very little organisational guidance and have an air of being somewhat anarchic. In addition, they work best for small development teams that work physically close together.

The OpenUP tries to combine both worlds: it takes inspiration from the Agile Manifesto (Beck et al., 2001) by incorporating the principles of agile methods such as customer involvement and embracing change, while documenting the process clearly and providing a complete set of guidances that allow a development team to design, implement, test, and deploy a software product based on best practices such as test-driven development and risk mitigation. It is well documented and provided as an easy-to-navigate website containing all necessary information and pointers to further reading. It adheres to standard current software engineering principles, e.g., by providing a risk-value based life-cycle (see Figure 13.3). The process itself as well as all accompanying material is available under an open source license and can thus be extended and altered freely. SPEM (cf. Section 12.1) is used as the description language of the process element. The Eclipse Foundation also provides a tool, the Eclipse Process Framework Composer that allows to browse and author processes—including, of course, the OpenUP. In fact, the process is part of the larger Eclipse Process Framework that provides the tools to create customised processes.

The Eclipse Process Framework contains a *practices library* in which a number of management and technical practices are defined that are then combined to yield the OpenUP. Artefacts, Tasks, Guidances, etc. that have been defined within these practices can be re-used and re-purposed. The management practices include *Iterative Development*, defining the iteration life-cycle shown in Figure 13.3 and corresponding tasks, work products and guidelines. Furthermore, the *Risk-Value life-cycle* on the project level is defined here as well by specifying the phases and milestones. The latter practice does not contain specific tasks, but rather provides a template and guidelines. Similarly, the *Whole Team* practice provides guidance on how a development team should be organised and how decisions are made within the team. Other practices, such as *Release Planning* and *Team Change Management* provide detailed tasks that have to be scheduled within the process. The practice library also defines a management practice for *Project Process Tailoring* that captures the tasks and a number of guidelines to adapt a process to the specific environment and circumstances of a concrete project.

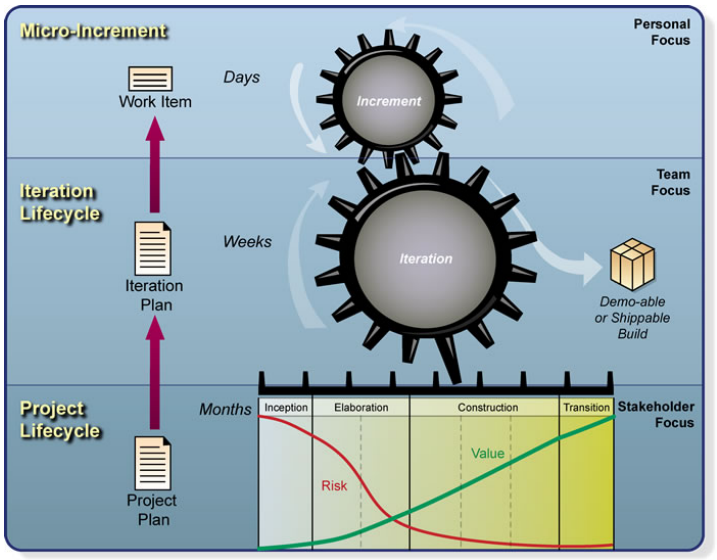


Figure 13.3: The life-cycle of the OpenUP (from `http://epf.eclipse.org/wikis/openup/index.htm`). The project is split into four phases which are, in turn, split into iterations. Within each iteration, a build of the software is created that can be tested by the customer or even shipped. The iterations are subdivided into increments in which relatively small, self-contained feature sets are implemented. During the process, value is continuously created by the addition of features while risk is mitigated by capturing requirements, designing, implementing, and testing solutions to the problems of the customer. Changes in these requirements can occur at any time and are incorporated in the design process as soon as possible.

The technical practices described in the practice library deal with all disciplines of the development process. In general, they are described on a very high level of abstraction. *Shared Vision* proposes to develop and communicate a vision of the product to be developed very early on in the process and to continuously validate the requirements with regard to their impact on this vision. The requirements discipline is further represented by the *Use Case Driven Development* practice which describes how use cases, system-wide requirements, and additional elements such as a glossary are created to capture the specification of the system. Analysis and design are represented by *Evolutionary Design* and *Evolutionary Architecture*. Strongly coupled with the iteration life-cycle, these practices suggest to incrementally create and refine the system architecture and design specific solutions for selected requirements in each iteration. These ideas are complemented by *Test Driven Development* and *Concurrent Testing*, two practices that promote the use of test cases to continuously check the design and implementation for consistency and validity. As an additional tool for the implementation and the test disciplines, *Continuous Integration* proposes to integrate and build the separate parts of the product often to detect and fix problems early. Deployment practices include *Documentation and Training* and *Release Planning*.

The EPF practice library also contains a "core" area in which common elements are defined. This includes categories, roles, work products (and work product slots), and guidance (especially common concepts). Noteworthy elements include the work product slots *[Technical Specification]*, *[Technical Architecture]*, and *[Technical Implementation]* that are refined by the practices in the PosoMAS practices library.

13.4 Structure and Content of the PosoMAS

The Process for Open Self-Organising Multi-Agent Systems follows the OpenUP in many regards. The main differences are the way requirements are handled and in the level of detail for the design activities. PosoMAS adds most of its method content in the latter area and replaces use cases as the main model to capture requirements with system goals. The following will briefly describe the combination of the practices from the EPF and the PosoMAS practices libraries before detailing the structure and life-cycle of the PosoMAS and outlining the connections and differences to OpenUP.

Combining EPF practices and PosoMAS practices

PosoMAS uses much of the existing content specified in the EPF practices library, replaces some of it, and adds content specific to open self-organising multi-agent systems where appropriate. This ensures that system designers that are familiar with the basic framework of iterative-incremental software development process will easily pick up the structure of the phases and iterations used in PosoMAS. The OpenUP's relatively generic descriptions of many parts of the software engineering process allow an

instantiation with more concrete content for specific cases while not having to re-invent generic content that is valid for all classes of systems that are created. For instance, the OpenUP defines a practice that describes the basic principles for iterative development that has been re-used without modification in PosoMAS. Likewise, management practices such as change management, involvement of the whole development team, or release planning are re-used.

EPF's technical practices, however, are very generic and have therefore been partially replaced in PosoMAS. This mainly concerns the practices "Evolutionary Architecture" and "Evolutionary Design". Their principles are subsumed in the more concrete "Evolutionary Agent Design" and the other design-centric practices. Furthermore, OpenUP prescribes use cases to model requirements and uses them in many of the design activities. PosoMAS replaces this with a goal-driven requirements elicitation approach and thus the practice "Use-case Driven Development" is replaced by "Goal-driven Requirements Elicitation".

The PosoMAS life-cycle and Work Breakdown Structure

PosoMAS adopts the OpenUP project and iteration life-cycle shown in Figure 13.3 by incorporating the EPF practices *Risk-Value life-cycle* and *Iterative Development*. Therefore, PosoMAS follows the OpenUP and divides the work in four phases as depicted in Figure 13.4. In each phase, specialised activities are applied to accommodate open self-organising multi-agent systems.

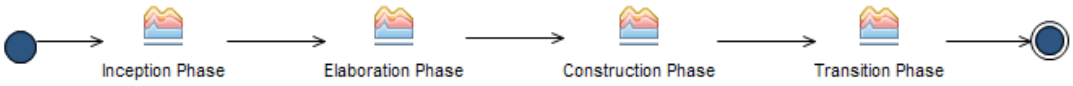


Figure 13.4: The phases of PosoMAS as defined by the risk-value life-cycle of the OpenUP.

The inception phase is often iterated only once and lays the foundational work for the project. During the elaboration phase, requirements elicitation, analysis, and design are the most important aspects and the software design matures. Construction is mainly concerned with implementation issues and it is assumed that the requirements have become relatively stable at that point. Release planning also begins. Finally, during the transition phase, the deployment of the product is prepared, final testing and documentation is performed and the product is rolled out. Elaboration, construction, and transition are usually iterated several times, according to the needs of the specific project.

Inception Phase During the initial iteration of the inception phase, the development team, the product owner, and the stakeholders have to come to an agreement about the scope of the project, including the features of the system and the final quality standards (task *Develop Technical Vision*, practice *Shared Vision*). For this purpose, extensive requirements elicitation work is performed in the beginning. In contrast to the use-case driven approach to requirements engineering promoted by the OpenUP, PosoMAS uses the goal-driven requirements elicitation from the practice of the same name. The requirements and the shared vision are also used to *Agree on a Technical Approach* (includes the task *Envision Architecture*, practice *Evolutionary Architecture*). Figure 13.5 shows the activity chart for the inception.

Practices Used
From the EPF practice library:
Shared Vision, *Release Planning*, *Evolutionary Architecture*, *Iterative Development*, *Project Process Tailoring*
From the PosoMAS practice library:
Goal-driven Requirements Elicitation

A noteworthy activity is the *Plan and Manage Iteration* activity as it is performed at the beginning of each iteration in every phase and since it differs from the inception phase to the other phases. During the inception phase, it includes a sub-activity *Prepare Environment*. Within *Prepare Environment*, a number of tasks from the EPF practice *Project Process Tailoring* are performed. They include tasks to define and deploy a customised software engineering process and prepare, set up, and validate the tools that support working with the process. Tools can include collaboration utilities, versioning software, continuous integration servers, as well as compilers and integrated development environments. In essence, this activity captures all steps necessary to define a specialised process for the team, the environment, and the product, and prepare the infrastructure necessary to start the development.

Plan and Manage Iteration also provides essential tools for the entire development team. For each iteration, an *Iteration Plan*, a *Risk List*, and a *Work Items List* are created. The iteration plan contains

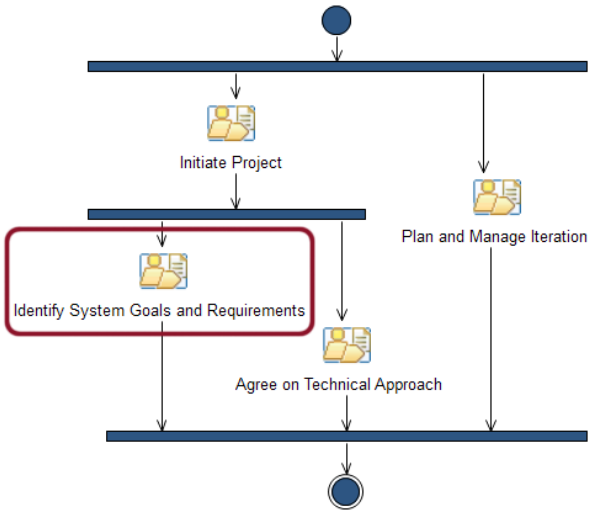


Figure 13.5: Overview of the inception phase of the PosoMAS. The activity *Initiate Project* is only executed once, even if several iterations are performed within the inception phase. *Plan and Manage Iteration* is a running task that is concurrently executed over the entire course of each iteration. *Identify System Goals and Requirements* and *Agree on Technical Approach* are also executed concurrently since the knowledge about the product that is required to create a technical approach only becomes apparent in the course of requirements elicitation. The activity that has changed in comparison to the OpenUP is indicated by the red frame.

the timetable for the iteration along with important events and milestones. Importantly, it also contains a list of issues that will be addressed during the iteration. These issues can be an important communication device since the development team can schedule any concerns that come up for discussion with the stakeholders, analysts, or the product owner by adding them to the iteration plan for the next iteration. In addition, a list of objectives is recorded in the *Iteration Plan*. When using a goal-driven requirements elicitation procedure, these objectives can correspond to sub-goals or individual requirements. The *Risk List* contains the critical points that still need to be addressed while the *Work Items List* provides a breakdown of the objectives to concrete work packages for the team members, along with an assessment of their complexity.

The inception phase of the PosoMAS follows the OpenUP closely, with the only notable difference being the replacement of the activity *Identify and Refine Requirements* for use-case driven requirements elicitation by the activity *Identify System Goals and Requirements* for goal-driven requirments elicitation. Just like in the OpenUP, the phase ends with a *life-cycle Objectices Milestone* that determines the project scope and the objectives the project has to fulfil at the end of the inception phase.

Elaboration Phase The elaboration phase puts the focus of the development team on the design of the software and the realisation of the requirements. At the same time, first implemented features generate value for the customer and are the basis for further elaboration of the requirements. The activity diagram in Figure 13.6 shows the most important activities. *Plan and Manage Iteration* is still performed at this point, although the sub-activity *Prepare Environment* is no longer a part of this activity. In addition to the activities already introduced, system design activities are now added. Early implementation and testing are also performed, along with change management.

The design activities include *Design Architecture Components*, *Design System Dynamics*, and *Develop Solution Increment*. In each iteration of the elaboration phase, a subset of requirements to work on is selected in *Plan and Manage Iteration*. The necessary changes to the design and a subsequent implementation are then performed to develop an increment that provides value to the customer and reduces the risk inherent in the project.

Design Architecture Components deals with the static parts of the system design and the trust infrastructure and includes sub-activities for the system architecture, the agent architecture, and trust-based interaction design. It also includes a task from *Model-driven Observer Synthesis* for the definition of the observation model. The use of patterns and other re-usable architectural elements is promoted by incorporating a corresponding task from *Pattern-driven MAS Design*.

Design System Dynamcis deals with the behaviour of the agents, their interactions, and agent organisations. Based on the requirements, the capabilities of the agents is identified and their behaviour is specified. The interactions between the agents are designed and interaction patterns and protocols are applied if possible. A suitable system organisation is selected and a *self-organisation* algorithm is specified if necessary.

Develop Solution Increment can be performed after these design activities have been completed. As test-driven development is practiced, developer tests are implemented before the solution code is produced. If all tests pass, the code is integrated and a build is created. Adherence to the system

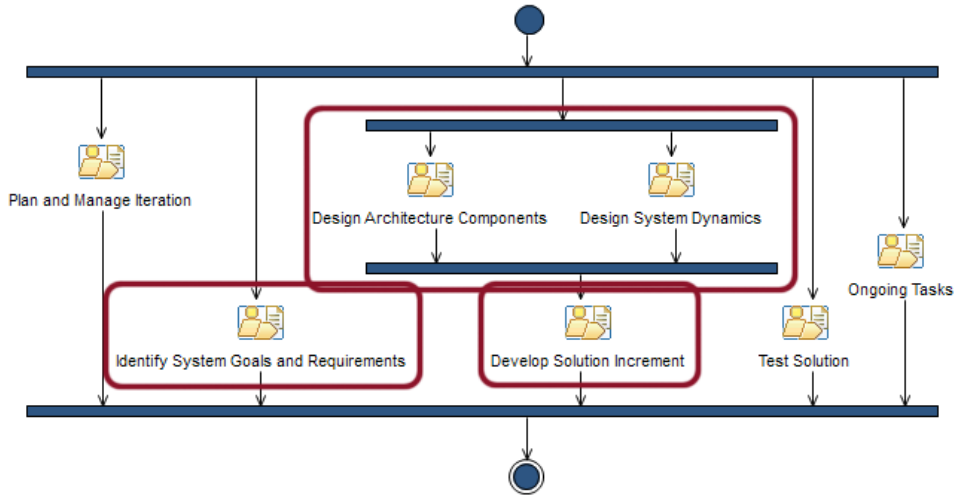


Figure 13.6: Overview of the elaboration phase of the PosoMAS. Iteration planning and requirements elicitation continues. In addition, a number of activities are included that deal with the design of the different aspects of the solution. *Design Architecture Components* covers the static parts of the product while *Design System Dynamics* includes tasks and activities for the system behaviour, interaction, and organisation. The design is implemented in appropriate activities and tested while code is generated. Change requests are handled within *Ongoing Tasks*. Elements that have changed in comparison to the OpenUP are indicated by red frames.

requirements, as well as component and integration testing is performed concurrently in the activity *Test Solution*. The transformation from the requirements to the observation model and from there to observer and controller code are also implemented at this point and the transformation is performed if changes in the models make it necessary. If any issues come up during implementation or testing, they are recorded in the iteration plan for discussion with the stakeholders at the beginning of the next iteration or captured in a change request.

Ongoing Tasks deals with the submission and integration of change requests. Any of the stakeholders of the project can issue a change request that will be reviewed and either accepted or rejected over the entire course of the project. Change requests can either influence the system requirements or the technical solution. In the former case, they are integrated into the requirements elicitation process and discussed with the product owner. In the latter case, they can influence the design of the solution or the implementation. In this case, they influence the development teams *Work Item List*.

Practices Used
From the EPF practice library: *Evolutionary Architecture*, *Iterative Development*, *Test-driven Development*, *Continuous Integration*, *Team Change Management*
From the PosoMAS practice library: *Goal-driven Requirements Elicitation*, *Evolutionary Agent Design*, *Trust-based Interaction Design*, *Model-driven Observer Synthesis*, *Pattern-driven MAS Design*, *Agent Organisation Design*

The elaboration phase of the PosoMAS differs from the one from OpenUP as the rather abstract *Develop the Architecture* activity has been substantiated with content from the practices *Evolutionary Agent Design*, *Trust-based Interaction Design*, *Model-driven Observer Synthesis*, *Pattern-driven MAS Design*, and *Agent Organisation Design*. The elaboration phase ends with the *life-cycle Architecture Milestone* that signifies that the most important aspects of the system, agent, and organisational architecture are completed. The most risky requirements have been tackled and appropriate solutions have been developed and incorporated into the design.

Construction Phase The construction phase marks a shift from design and requirements elicitation towards implementation and preparation for an eventual release. While the overall structure of an iteration within this phase is similar to the structure of an iteration in the elaboration phase, a number of tasks that have been performed during elaboration are no longer performed. In addition, preparatory release and documentation activities are introduced. Figure 13.7 shows the most important activities and the order in which they are performed.

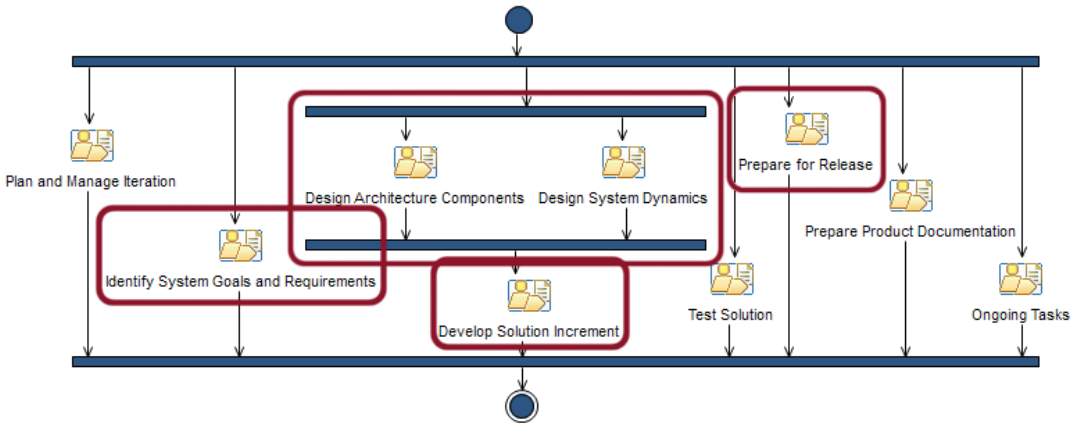


Figure 13.7: Overview of the construction phase of the PosoMAS. Iteration planning and requirements elicitation continues. The design activities are still present but many of the tasks they contained in the elaboration phase are no longer performed. The existing design is implemented in appropriate activities and tested while code is generated. Change requests are handled within *Ongoing Tasks*. Additionally, preparations for the final release begin and training documents and documentation are created. Elements that have changed in comparison to the OpenUP are indicated by red frames.

Since the design activities are assumed to be mostly finished and the system, agent, organisation, and interaction design should have stabilised, a number of tasks are no longer performed. Most prominently, tasks from the practice *Pattern-driven MAS Design* are no longer part of the design activities since the design should be stable enough so that relevant architectural patterns, protocols, etc. already have become part of it. The observation model should also have stabilised, sources of uncertainty relevant for trust models have been identified, the agent capabilities have been defined and the suitable agent organisation has been selected. This leaves tasks concerned with the refinement of the existing architectural components.

Practices Used
From the EPF practice library:
Evolutionary Architecture, *Iterative Development*, *Test-driven Development*, *Continuous Integration*, *Team Change Management*, *Documentation and Training*, *Production Release*
From the PosoMAS practice library:
Goal-driven Requirements Elicitation, *Evolutionary Agent Design*, *Trust-based Interaction Design*, *Model-driven Observer Synthesis*, *Agent Organisation Design*

As the product matures, *Prepare for Release* becomes part of the iterations. This activity is mainly concerned with the development of a release plan, identification and procurement of the relevant infrastructure, as well as the creation of a backout plan that allows the team to rollback the deployment in case something goes wrong. In addition, a notification for all stakeholders is prepared that details the changes and raises awareness for possible problems within the group of users. As open self-organising multi-agent systems are potentially deployed on a complex, distributed infrastructure, this activity is tackled relatively early in the development process to integrate feedback from this step in the system design if necessary and to mitigate the risks that could potentially arise in the deployment as early as possible.

In parallel to these activities, *Prepare Product Documentation* is performed. The activity contains task for the creation of user documentation, product documentation, support documentation, as well as training material. The documents created as part of the development effort can serve as starting points for many of these documents.

The construction phase of the PosoMAS differs from the one from OpenUP again in the more specialised design activities and in incorporating early release planning for the reasons outlined above. The construction phase ends with the *Initial Operational Capability Milestone*, an extended prototype that is usable as a standalone product. It contains the most important functionality and thus covers the most important requirements. Testing is mostly finished and a preliminary product documentation is available. In addition, plans for deployment have been defined.

Transition Phase The final iterations of the process are part of the transition phase in which development is wrapped up and a final release is created and deployed. Design activities are suspended at this point as the overall design is completed. Minor details and fixes can still be implemented and tested. Change requests can no longer result in new requirements or changes in the design but have to

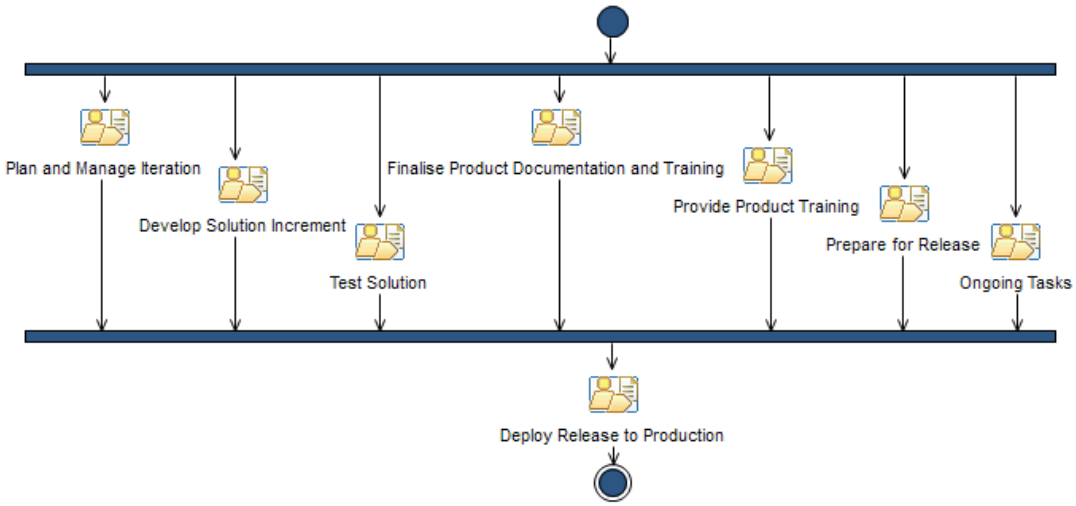


Figure 13.8: Overview of the transition phase of the PosoMAS. Iteration planning continues. The requirements elicitation as well as the design activities are no longer relevant. Final implementation and testing can still be performed. Change requests that do not impact the design are handled within *Ongoing Tasks*. Additionally, preparations for the final release continue and training documents and documentation are finalised. When all tasks have been performed, the release is deployed.

be realised on the code level. The product documentation and the training documents are finished and product training starts for all relevant stakeholders. The release preparations are completed and the final release is deployed. Figure 13.8 shows the overall structure of an iteration in the transition phase.

Provide Product Training addresses both the end users and support staff. Depending on the scope of the project, a limited group of users can be identified and instructed in the use of the software. In case of open self-organising MAS, this group is potentially very large and can not be approached by the development organisation easily. In such cases, the training can be provided to specialised staff that will in turn train the end users in the field. Support staff is instructed in more technical detail. The documentation created during the corresponding activities and during the development effort can be used.

Practices Used
From the EPF practice library: *Iterative Development*, *Test-driven Development*, *Continuous Integration*, *Team Change Management*, *Documentation and Training*, *Production Release*
From the PosoMAS practice library: None

Deploy Release to Production is the final step in the process. The release plan prepared earlier is executed, a release is packaged, deployed, the release is verified, and the release is communicated to the stakeholders. If necessary, the backout plan is executed. In case of open self-organising multi-agent systems, the release can be rolled out in phases. In such a case, specialised staff that supervises the deployment can be trained and put in place.

The transition phase of the PosoMAS does not significantly differ from the one in OpenUP. All activities specific to open self-organising MAS have been completed in the previous phases. Documentation and deployment activities are not fundamentally different from a traditional software product. The transition phase ends with the *Product Release Milestone* for which the stakeholders have accepted the final product, training and documentation is completed, and deployment has been successful.

13.5 Addressing the Requirements for an AOSE Methodology for the System Class

The requirements for which PosoMAS as a new AOSE methodology has been created are outlined in Section 12.3 and the way these requirements have been addressed is described below. The literature provides a more exhaustive evaluation framework which is used in the validation of the process in Chapter 14. This validation also substantiates the claims that the requirements have been addressed sufficiently.

Accessibility to "traditional" software engineers: PosoMAS promotes accessibility by, on the one hand, using a well-known software engineering process life-cycle and embedding new method

content into an existing process framework and, on the other hand, by providing a plethora of guidance to help a software engineer unfamiliar with agent-oriented designs overcome the barriers to entry a new paradigm can present.

Architecture and tool agnostic: The process does not contain specific content that would make it necessary to decide for an agent architectural paradigm such as BDI. It promotes the use of standard modelling languages and thus allows the use of any CASE[2] tool that uses UML.

Level of detail: The four architecture areas guided the creation of respective practices within the PosoMAS practices library. The relevant design documents are thus created in the tasks contained in these practices. The guidance provided within the practices can serve as a starting point for an in-depth induction in the area and detailed descriptions allow the software engineer to understand the scope and purpose of the different tasks. The PosoMAS through its combination with the OpenUP covers the entire life-cycle of a modern software engineering methodology.

Extensibility and customisability: As witnessed by the combination with the OpenUP and with Scrum in Section 13.6, the method content can easily be combined and extended. Customisations can be applied by using standard tools. The practices are available in a repository and can be re-used due to their non-restrictive license.

Clear separation of different architecture domains: PosoMAS provides a number of specialised practices, tasks, activities and work product slots for the four architecture domains agent architecture, agent interaction, agent organisations, and system architecture. The process promotes a concurrent design while clearly separating the domains and the corresponding models and documentation. The requirements elicitation practice also advocates the practice of precisely delineating the requirements for the different domains.

Special focus on interactions between agents and agent organisations: Interactions of agents are covered in the practices *Evolutionary Agent Design* and *Trust-based Interaction Design* as well as in *Pattern-driven MAS Design*. The corresponding tasks allow tailoring the process for the required level of detail for the agent interactions, e.g., by omitting *Trust-based Interaction Design* and promotes reuse of existing protocols and communication standards. *Agent Organisation Design* is covered in a separate practice and includes identification of a suitable organisation as well as the specification of a self-organisation algorithm. Interactions and the agent organisation are captured in respective work product slots.

Top-down design vs. bottom-up functionality: PosoMAS promotes the use of methods for goal-driven requirements elicitation that break down system goals to fundamental requirements on the micro-level. These requirements have to be fulfilled by the individual agents, possibly through a self-organisation process. The specification of such a process is part of the practices as well. The combination of individual, heterogeneous agent architectures and requirements, an overall system perspective with respective goals and architectural elements, and the use of self-organisation thus provide both top-down and bottom-up perspectives.

13.6 Customisation of the Methodology

Based on the description of the process in the previous section, we now outline how the process can be customised and tailored to specific projects, teams, and environments. This includes an identification of the possibilities for process customisation, the project requirements under which an adaptation can be necessary, practices and method fragments from other processes that can be integrated, and the adoption of more heavy-weight procedures for documentation. An outline of how PosoMAS could be integrated with Scrum is also provided.

Customisation for project-specific technology decisions

For each system implemented with PosoMAS, the designer has to define the agent concept and the agent's reasoning technique (BDI, rule-based, expert-system based, etc). These decision can be facilitated by specialised practices for different agent modelling approaches such as BDI. Appropriate method fragments or chunks from other processes, e.g., those presented in Chapter 12, can be combined with the practices presented here to achieve this. Presently, however, method fragments or method chunks

[2]Computer-aided Software Engineering

from most other AOSE methodologies are not available in a reusable form. Such a combination is thus a subject of future work and collaboration with the creators of the methodologies.

To demonstrate the possibility of such a combination, we define a practice based on the method fragment *Adequacy Verification* from ADELFE (Cossentino and Seidita, 2005). The original description is very brief, but (Bernon et al., 2011) provides some more information about the tasks and the guidelines an analyst has to follow. The method content is defined within a practice in the EPF Composer in a separate method plugin. This allows us to use roles, work products, and other content defined within the EPF practices library and the PosoMAS practices library. Descriptions are partially copied from the papers, partially newly written. The practice consists of the task *Confirm MAS Adequacy* with four steps, the guidance *MAS Adequacy Criteria* that contains a list of criteria to decide whether implementing as a MAS is in order defined in (Bernon et al., 2011), and a work product *MAS Adequacy Synthesis* as defined in the fragment description in (Cossentino and Seidita, 2005). The work product is produced by the aforementioned task which takes a *[Technical Specification]* as the basis for the decision since some architectural elements must be known to evaluate the criteria.

The task has been assigned to the *Requirements* discipline since it should be performed very early on in the project and only requires a preliminary technical specification that is, e.g., provided by the *Conceptual Domain Model* created by the *Goal-driven Requirements Elicitation* practice. Therefore, the practice, in particular the single task defined in it can be integrated into the inception phase as depicted in Figure 13.9. The initial requirements are elicited first before the MAS adequacy is confirmed. In parallel, the technical approach is agreed upon. This activity contains the task *Envision the architecture* from the EPF practice *Evolutionary Architecture* and is useful to define an initial framework in which the software architecture can be developed. Since both tasks require architectural requirements and since some of the criteria for MAS adequacy can best be answered in the context of a system architecture, they are executed in parallel.

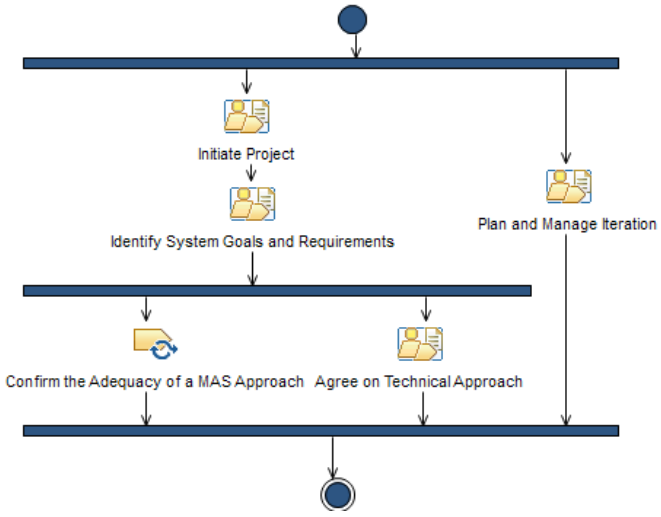


Figure 13.9: The inception phase of PosoMAS extended with the practice *MAS Adequacy Confirmation* from ADELFE. After the initial requirements are determined, a vision of the system architecture is developed and in parallel, the adequacy of a multi-agent system approach is determined. If the architectural analysis shows, e.g., that a large number of components, interacting in a non-linear fashion in a highly dynamic environment is needed, the criteria point towards the adequacy of MAS. The result of the confirmation is recorded in the *MAS Adequacy Synthesis*.

The confirmation of MAS adequacy is a potential showstopper for using PosoMAS. If the adequacy of multi-agent systems can not be confirmed and the development team instead decides to create a "traditional" software system, the selected process is no longer applicable. Instead, a new process has to be selected that is compatible with the product.

A different customisation applies if a certain implementation framework is selected. If, e.g., a multi-agent implementation framework that provides facilities for BDI agents is selected, the process should be augmented with method content that aids the architect and the development team in the definition of beliefs, desires, and intentions. In such a case, method content from specialised processes such as Prometheus (Padgham and Winikoff, 2005) can be used. Prometheus offers guidance during its *Detailed Design* phase to successively define and refine goals, events, and plans to be used by the agents to reason about next steps and perform actions in their environment. This method content can be converted into practices for reuse in PosoMAS or other processes. The main difficulty in such a conversion is to ensure compatibility of the work products. Work product slots can be helpful in the syntactical matching, but semantics are important as well. Potentially, the process tailoring step thus has to accommodate new method content accordingly and adapt work products so they are interchangeable between practices and tasks.

Customisation for Team Size and Experience

Team size and experience are important factors in the customisation of a process and its deployment. Agile processes are best suited for small teams since they focus on quick decisions and communication. Extensions for larger team sizes exist in some processes. Scrum, e.g., compartmentalises the effort and introduces a "Scrum of Scrums" in which designated members from different teams perform a separate daily scrum. The OpenUP used as the basis for PosoMAS does not prescribe a certain team size but due to its heritage as a descendant of the Rational Unified Process is fine-grained enough and based on sufficient documentation to make it work in larger teams in which communication is not ubiquitous.

OpenUP (and thus PosoMAS) defines a relatively large number of different roles. In small teams, some of the roles defined in the PosoMAS must be filled by the same member. Larger teams might have specialist deployment engineers or testers, but in smaller teams such roles are usually taken on by the developers themselves. Similarly, the architect role is often filled by a subset of the members of the regular development team. Such an approach has advantages since more people are "in the know" and less information has to be communicated through diagrams and documents. On the other hand, the team has to devote time to these tasks that can not be spend fulfilling their main role. If the team is larger, more work can be performed in parallel but more effort must be spend to communicate important decisions. Team size is always a trade-off between these factors but PosoMAS supports flexible role assignment and is thus able to handle both small and larger teams.

Customisations for an inexperienced team mostly deal with the length and number of iterations. In general, an inexperienced team should at least have an experienced project lead. Since multi-agent systems have not permeated mainstream software engineering yet, it is to be expected that most development teams will be new to developing a MAS. A project manager who can at the same time serve as a technical guide is therefore a necessity. To bring an inexperienced team up to speed, the initial iterations should allow time for the developers to get acquainted with the technology and the theory behind it. This will take time away from the other tasks necessary at the start but the effort will pay off later on. To allow the team to familiarise themselves with multi-agent systems, it is also advisable to keep the initial iterations of the elaboration phase relatively short. This implies a small number of requirements tackled in each iteration and product increments that are available often. Feedback can thus be provided quickly and for a relatively small amount of work, making it easier to pinpoint mistakes and giving the team opportunities to learn from them. Once the development team develops the necessary understanding and skills, the iterations can be set to their regular length of a few weeks.

Customisation for Scrum

While the OpenUP as defined by the EPF community is an agile process in many aspects, it still contains the foundations of a strict, waterfall-like process. This is evident in the differentiation of four phases and in the emphasis on documentation. Scrum is an example of a light-weight process framework that embraces the Agile Manifesto (Beck et al., 2001) more wholeheartedly. It consists of a number of rules that prescribe how roles, events, and artefacts have to be combined for a Scrum Team to manage a complex software development project and create value for the customer (Schwaber and Sutherland, 2011). Within these boundaries it is possible to apply custom development practices. Scrum has been modelled in the EPF Composer by Mountain Goat Software[3] and a method plugin containing the method content is available. The customisation is based on this method content.

Structure of Scrum At the core of Scrum is the insight that it is impossible to conclusively define requirements and that it is therefore necessary to include the client in all phases of the development and be able to react to changes in the requirements quickly. A change request is thus not an exceptional event but something rather normal, making it very easy to incorporate changes into the development process. The decisions of the project team are always based on what is currently known about the project, a principle called *empiricism* (Schwaber and Sutherland, 2011). This makes it necessary to make decisions transparent for the stakeholders, inspect past decisions based on new data regularly, and adapt if the circumstances have changed.

The other principle is the focus on a self-organising development team. While different roles still exist, Scrum promotes the notion that different members of the development teams assume these roles of their own accord. The team decides internally how the work is split among the team members and is

[3]The corresponding content can be found in a human-readable form at `http://epf.eclipse.org/wikis/scrum/`.

involved in all important decisions, including assessment of the effort required to perform certain tasks. A *Scrum Master* is designated in each team that, on the one hand, ensures that the Scrum rules are adhered to, but also acts as a spokesperson for the development team and coordinates communication with external stakeholders. Importantly, a *Product Owner*, ideally a representative of the client with the authority to make decisions about the product and embedded in the development team, defines the requirements and orders them for importance. She is also available to the development teams to answer questions and relay issues that come up during development to the client organisations. The Product Owner has, however, no authority over the development team (Schwaber and Sutherland, 2011).

The most important communication device between the development team and the product owner is the *Product Backlog*. It consists all requirements for the product, including features and non-functional requirements (Schwaber and Sutherland, 2011). While Scrum does not prescribe the form in which requirements are represented in the product backlog, they are often captured in the form of *user stories* (see, e.g., Moore et al., 2007). A user story describes *who* wants to achieve *what* and *why*. The stories are ordered in the backlog according to their risk, customer value, complexity, or any other criteria the product owner deems reasonable.

The development team estimates the effort required to realise the user stories. As part of a "grooming" process, the product owner and the development team together refine the stories, re-order them and adapt estimates. As part of a *Sprint Planning Meeting*, the user stories that will be tackled in the next development iteration—called a *Sprint*—are picked and transferred into the *Sprint Backlog*. During the sprint, the development team creates a complete product that realises the picked user stories (the *Sprint Goal*) over a time horizon of a month. The product owner can cancel the sprint if the sprint goal no longer aligns with the client organisation goals. A sprint is different from an iteration in the OpenUP as it is not assigned to a specific phase. Instead, all necessary work to deliver the sprint goal is performed during a sprint with no discernible shift of focus in later sprints.

During a sprint, a number of strictly structured meetings are performed to further communication between the development team members and to assess the progress of the sprint. The *Daily Scrum*—also called the *Standup Meeting* since it is usually held with all participants standing—is a 15-minute meeting in which the progress, the planned work, and any issues can be discussed (Schwaber and Sutherland, 2011). In larger projects, a *Scrum of Scrums* can be used with similar rules to coordinate between different development teams. Scrum promotes co-location of all team-members, meaning that it is easy to communicate directly with other members of the team, to exchange information, and to help each other with technical difficulties.

Embedding PosoMAS practices in Scrum The description above does not include concrete development practices. Indeed, Scrum—similar to the OpenUP—is seen as a framework, defining rules and an environment, in which concrete development practices can be applied. The EPF Scrum Plugin does not include a process life-cycle in which the method content can be embedded. Therefore, we use a variant of Ambler's (2012) "Agile System Development life-cycle (SDLC)", a refinement of the standard Scrum life-cycle. It contains a framework for the individual sprints, including project setup and initial requirements and deployment and operation. Sprints are embedded in iterations which contain sprint planning and review. The version of the life-cycle used here is shown in Figure 13.10.

The PosoMAS practices are used within a sprint. A sprint, depicted in Figure 13.11 contains the activities defined in the PosoMAS practices library. The development team decides which activities, sub-activities and tasks it has to tackle each given day. This gives them a flexible structure within which to work. All guidances, work products, and other method content defined in the PosoMAS practices are at their disposal. In general, Scrum puts less focus on documentation. Therefore, a development team can decide to create less models or combine models suggested by PosoMAS activities and tasks. Since most of the management documents has been defined in the OpenUP and is not present in the Scrum/PosoMAS method configuration, no additional customisation has to be performed in this regard.

Notably, there are no requirements engineering activities in the sprint breakdown. This is due to the fact that scrum works with a backlog and user stories as mentioned above. The initial requirements are captured in the release planning stage, before the sprints are begun, a task similar to the *Initiate Project* activity described for the OpenUP earlier. The product backlog is refined by new user stories that are created during sprints or during sprint planning meetings. Therefore, requirements elicitation is a continuous process but is not using either the use-case-based method proposed in the OpenUP nor the goal-oriented one put forward by PosoMAS.

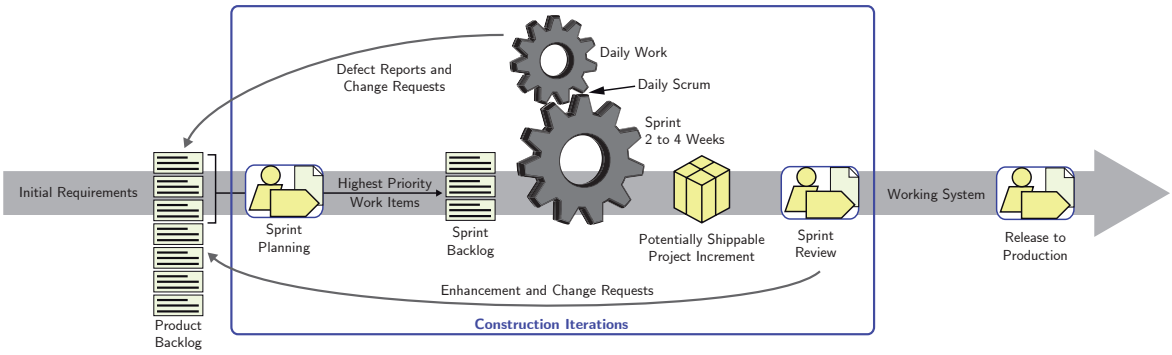


Figure 13.10: The life-cycle of the PosoMAS variant of Scrum. Initial requirements are captured in a *Product Backlog*. An iteration consists of *Sprint Planning*, in which the backlog is prioritised and the work items for the sprint are selected. The sprint itself consists of small increments of *Daily Work*, initiated by a *Daily Scrum*. After the 2 to 4 weeks of the sprint are completed, a *Potentially Shippable Project Increment* has been created. The *Sprint Review* gives all stakeholders the opportunity for feedback. After construction is complete, the working system is released to production. The product backlog can be changed or augmented by the stakeholders at any time during or after a sprint. Diagram based on the "Agile System Development life-cycle (SDLC)" (Ambler, 2012), an extension of the standard Scrum sprint lifecycle with iterations and consideration of preparatory as well as release activities.

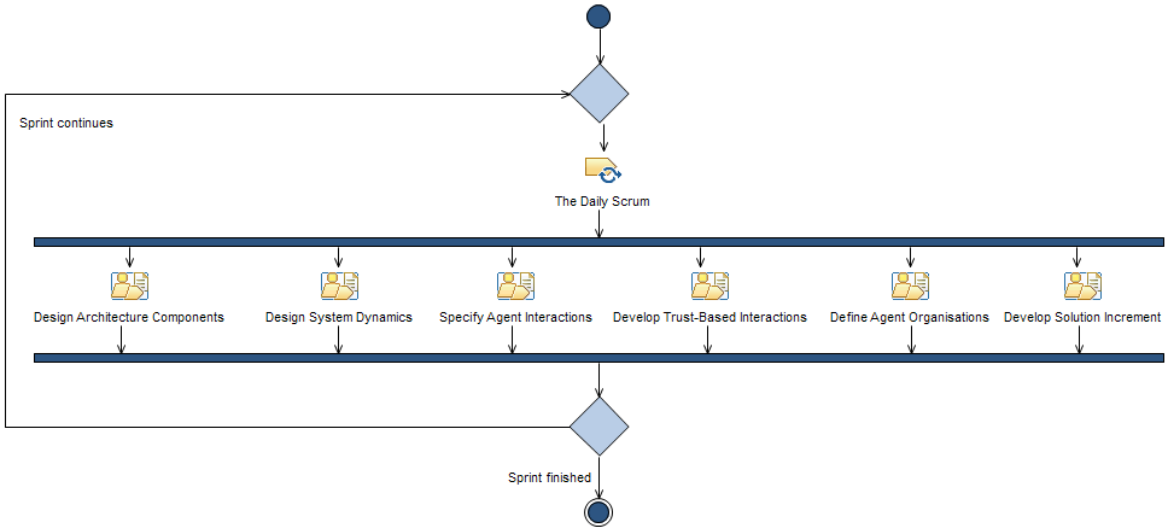


Figure 13.11: Structure of a sprint in the PosoMAS variant of Scrum. Each day is kicked off with a daily scrum. The activities from PosoMAS are performed as required during the daily work. The sprint ends after a certain time or when all backlog items have been tackled.

Chapter Summary and Outlook

After defining the notation used for the elements of the agent-oriented software engineering methodology for large-scale open self-organising systems, the practices that make up the process are described. They cover important aspects of the system class such as hierarchical system structures, observation and control infrastructure, and trust-based interaction design. At the same time, characteristics that provide help in the engineering of large-scale multi-agent systems, including a strong differentiation between the different levels of observation, are established as cross-cutting concerns of different practices. The use of the method content in the context of the Open Unified Process defines a new process—PosoMAS—which is analysed with regard to the requirements put forward in Chapter 12. Further customisations, e.g., for Scrum are shown as well.

Chapter 14

Validation of PosoMAS with Two Case Studies

Summary. The applicability of PosoMAS to agent-based software projects is validated by applying the process to two case studies and evaluated according to a number of criteria that treat process-related, model-related, technique-related and supportive aspects. The work products created while applying PosoMAS to the case studies and the work flow in an actual project are described. The process customisations applied to the process for the different case studies as well as the lessons learned in the course of the validation are discussed as well. Based on a catalogue of evaluation criteria, PosoMAS is also characterised in comparison to other agent-oriented software engineering methodologies.

As laid out in Section 12.3, the validation of a software engineering process is difficult from a methodical point of view. Ideally, the process is tested in a productive environment for the creation of a software product with an experienced team of software engineers and developers who can then compare the effort to previous experiences with other methodologies. As argued in that same section, such an approach is not feasible in the scope in which AOSE methodologies are created at the moment.

Instead, we rely on simulated development efforts and on qualitative evaluation and validation criteria. Tran and Low (2005) have introduced a catalogue of criteria that will be introduced in Section 14.1 and used in Section 14.4 to show the strengths and weaknesses of PosoMAS and to compare it to other approaches. The basis for these evaluations are simulated development efforts for the case study on self-organising emergency response systems (So-ERS) and for the power management case study, the results of which and some interesting aspects are shown in Section 14.2 and Section 14.3 respectively. The way PosoMAS has been tailored for the case studies will be described in the respective sections as well as part of the *Project Process Tailoring* activities in the respective inception phases.

The scenarios have been chosen since they differ in a number of factors. The So-ERS is a highly connected system that includes sensors, information retrieval and distribution, as well as a pronounced human component. The power management case study on the other hand puts the focus on self-organisation and self-optimisation in a fully autonomous system. This diversity allows us to demonstrate that PosoMAS is not restricted to a small domain but applicable to a wide range of open multi-agent systems if tailored appropriately.

Both simulated process runs are based on the same course of action. The process runs have been performed by two relatively inexperienced students under our guidance. They were executed as defined in their description, with an inception phase, followed by elaboration and construction. The transition phase has not been performed since PosoMAS does not add additional content to the OpenUP for this final phase yet. Each iteration was planned with an *Iteration Plan*, an artefact detailing the work that will be conducted during the iteration. A *Work Items List* and a *Risk List* have been maintained as well. The overall project was laid out in a *Project Plan* but the original timetable was unachievable. This fact can be ascribed to the relative inexperience of the students who performed the process and the corresponding delays in reaching important milestones, especially w.r.t. a suitable system design.

The description of the process runs is structured according to the phases and to the activities performed in each of the phases. Within the activities, method content from different practices is often combined. Relevant work products are shown and the way guidelines and additional information material has been used is detailed. Based on the experiences of the evaluators, the criteria are appraised in Section 14.4, guidance for the selection of a process is provided and lessons learned are discussed.

14.1 Evaluation and Validation Criteria for Agent-oriented Software Engineering Methodologies

As outlined in Section 12.4, no validation method for software engineering processes is flawless. Personal preferences and biases, familiarity with the evaluated processes, learning effects, prior experiences, etc. always prevent a true objective evaluation. Nonetheless, it is crucial to provide evidence that a process is applicable and give guidance as to its specialities. We follow Tran and Low (2005) and Tran et al. (2005) by evaluating PosoMAS in four areas: process-related criteria, including aspects that are related to the process life-cycle, the steps that are used, and the agent architectures that are supported; technique-related criteria, evaluating the techniques employed in the individual model steps; model-related criteria, looking at the notational aspects and the concepts included in the models used; and supportive-feature criteria, focusing on tools and support for various AOSE specialities. We will not use the full catalogue used in the original paper and instead incorporate some of the aspects from (Al-Hashel et al., 2007) to bolster the structural evaluation.

The criteria put forward in (Tran and Low, 2005) and (Tran et al., 2005) are, at times, very specific. They contain, e.g., criteria such as "*Communication ability*: Can the models support and represent a 'knowledge-level' communication ability (i.e., the ability to communicate with other agents with language resembling human-like speech acts)?" (emphasis in the original text). From this follows that the evaluation is mostly tailored to methodologies that provide specialised models instead of re-using standard software engineering modelling techniques the way PosoMAS does. Other criteria explicitly evaluate the use of "standard AOSE concepts", e.g., roles for agents or the use of ontologies. For reasons outlined in Section 12.3, such elements are explicitly ignored in PosoMAS, mainly to avoid limiting the applicability of the process to specialised architectures or agent meta-models. Therefore, some of the model-related criteria, e.g., the one mentioned above or "*Human Computer Interaction*: Do the models represent human users and the user interface?" are ignored or altered in the following, since they definitely can but PosoMAS does not prescribe how or if this has to be done. The technique-related criteria and criteria that have to do with the steps of the process are regarded separately as well.

On the other hand, the criteria used by Tran and Low do not always provide clear guidelines on how they have to be evaluated. The criterion "complexity", e.g., is defined as "is there a manageable number of concepts expressed in each model/diagram?" Clearly, the definition of "manageable" differs quite a bit between different evaluators. In addition, the authors use some terms in the description of the criteria that are commonly employed differently in the SE community. Analysis, design, and implementation are usually not designated as the "phases" of a process but as its disciplines. These oversights have been fixed as far as possible in the overview below. If not explicitly denoted otherwise, all criteria and their description are from (Tran et al., 2005).

Process-related criteria These criteria are used to determine general properties of the process such as the supported life-cycle as well as its specific suitability for multi-agent system development, especially for the kind of MAS it can be applied to. The criteria are listed in Table 14.1. They can be assessed based on the process description, especially the life-cycle and the activities that are defined. A criterion not present in (Tran et al., 2005) is "meta-model based". A meta-model based process prescribes a meta-model for the description of the agent and/or system architecture. As argued in Section 12.1, meta-models can be helpful but they also limit the designers freedom in design choices.

Criterion	Description
Development life-cycle	Is the life-cycle formally defined? Which life-cycle is adopted (e.g., Waterfall)?
Coverage of the life-cycle	What disciplines are covered by the methodology (e.g., analysis, design, implementation)?[1]
Development perspectives	Does the methodology support both bottom-up and top-down development or does it follow a hybrid approach?
Application Domain	Can the methodology be applied to arbitrary domains or is it suited for specific domains?
Size of MAS	Which system size does the methodology support?

[1] Was originally: "What phases of the life-cycle..."

Agent paradigm	Is the methodology suited for the development of agents of a specific paradigm (e.g., BDI agents) or is it independent of the reasoning mechanism and meta-model used?[2]
Support for verification	Are there provisions for the validation and verification of the models and specifications developed in the methodology?
Refinability	Does the process define how a model is gradually refined and augmented in the course of the process?
Approach towards MAS development	Does the methodology apply a specific MAS development approach (e.g., based on methods form knowledge engineering, object-oriented, role-based, or "non-role-based", i.e., relying on other means such as use cases, workflow models, or interactions)?
Meta-model Based	Does the process prescribe a meta-model as the basis for architectural models of the system?

Table 14.1: Process-related criteria for the evaluation of agent-oriented software engineering processes according to Tran et al. (2005)

Model-related criteria The models created in the course of a process are the core vehicle to document design decisions, communicate to the stakeholders, guide the developers, and determine the scope of the project. Therefore, a number of criteria evaluate their expressiveness, the concepts they capture, and the way they are used in the process. As noted above, some of the criteria originally defined by Tran et al. are omitted or changed, since they were either formulated with very specific expectations about a MAS in mind or are based on the discretion of the designer. The "complexity" criterion, e.g., can be easily mitigated by creating different diagrams for a model, thus providing different views on the same model. While the model can be highly complex, the views—usually different diagrams—can shed light on specific aspects and thus simplify its depiction tremendously. The criteria are listed in Table 14.2. Tran et al. also provide a list of MAS concepts that are used to evaluate the coverage of more specific notions. This list is used in Section 14.4 as well.

Criterion	Description
Syntax and Semantics	Are the syntactical elements of the models and their semantics clearly defined?[3]
Model transformations	Is the transformation of models as part of a model-driven engineering approach supported by providing guidelines for the transformations?[4]
Consistency	Do provisions exist that guarantee internal consistency within a model and between models? Is it possible to check consistency between levels of abstractions and between different representations of the same aspect in different models?
Modularity	Can agents be structured in a modular way?
Abstraction	Is it possible to model the system and the agents at different levels of detail and abstraction?
Autonomy	Does the methodology support modelling the autonomous features of an agent?
Adaptability	Are features of adaptability such as learning supported by the models?
Cooperative behaviour	Does the methodology support modelling the cooperative behaviour of the agents?
Inferential capability	Is it possible to model automatic agent inference, i.e., the capability of an agent to derive concrete actions from abstract commands?

[2]Was originally: "Agent nature".
[3]Was originally: "Formalization/Preciseness of models"
[4]Was originally: "Model derivation"

Criterion	Description
Communication ability	Does the methodology support modelling communication and knowledge exchange between agents?
Personality	Does the methodology support modelling an agent's "ability to manifest attributes of a 'believable' human character"?
Reactivity	Is there support for the modelling of reactive agents that act as a response to sensed stimuli?
Proactivity	Does the methodology support modelling proactive agent behaviour, i.e., the ability of the agents to self-initiate a deliberately act to achieve a certain goal?[5]
Temporal continuity	"Can the models support and represent temporal continuity of agents (i.e., persistence of identity and state over long periods of time)?"
Concurrency	Do the models prescribed by the methodology allow capturing concurrency of and synchronisation between processes?
Model Reuse	Is is possible to reuse models from a library and is such a library provided with the methodology?

Table 14.2: Model-related criteria for the evaluation of agent-oriented software engineering processes according to Tran et al. (2005)

As PosoMAS uses standard UML diagrams with a specialised UML profile, most of the criteria about the expressiveness of the models can be answered in the positive. The question becomes whether the methodology gives guidance how these features can be modelled with standard UML. While some of the criteria are very general and apply to all kinds of multi-agent systems, some aim at very specific kinds of MAS or agents. The "personality" criterion, e.g., is applicable when an agent should act as a placeholder for a human. If a game is developed, the non-player characters could have a "personality" of their own. The role of emotion has been investigated in the multi-agent systems community as well (see, e.g., Brave et al., 2005) and personality and emotion can be linked (Allbeck and Badler, 2002). However, this aspect of multi-agent systems is usually ignored for AOSE processes since they are mostly aimed at technical solutions without strong requirements for such concepts.

Supportive feature criteria An AOSE methodology can support the development team in a number of ways that are captured in this set of criteria detailed in Table 14.3. This includes special features such as mobile agents or ontologies but also support for a number of other properties, such as self-organisation and self-interested agents. These criteria thus include a number of important principles for open self-organising MAS. As the evaluation will show, these criteria also capture the most important differences of the processes. This can be ascribed to the fact that Tran et al. added a number of central concepts to these "supportive features". The criterion *Dynamic Structure*, e.g., deals with self-organisation, an important part of processes such as O-MaSE and PosoMAS.

Criterion	Description
Software and methodological support	Are there tools and libraries of reusable components that support the methodology?
Open systems and scalability	Does the methodology provide support for open systems that allow agents and resources to dynamically enter and leave the system?
Dynamic structure	Is there support for self-organisation processes, i.e., the dynamic reconfiguration of the system structure?
Performance and robustness	Are techniques for dealing with exceptions, capturing error states and recovering from failures as well as performance monitoring provided?[6]
Support for conventional objects	Can "ordinary objects" be integrated into the design and are interfaces to such objects captured?

[5] Was originally: "Deliberative behaviour"
[6] Was originally: "Agility and robustness"

Criterion	Description
Support for mobile agents	Are there possibilities to integrate mobile agents in the developed MAS?
Support for self-interested agents	Is there support for agents that do not adhere to the benevolence assumption and thus do not necessarily contribute to the overall system goal?
Support for ontologies	Can ontologies be integrated in the design of the MAS?

Table 14.3: Supportive feature criteria for the evaluation of agent-oriented software engineering processes according to Tran et al. (2005)

Process Steps and Related Criteria In addition, Tran and Low propose 19 crucial process steps by analysing existing methodologies. The importance of these steps has been verified by "experts" according to the authors and can thus be seen as an agreed-upon minimal set of steps for the design of a multi-agent system. The presence of these steps, the "notational components" that are created in these steps, the weaknesses and strengths of the concrete realisations, the ease of understanding, and the definition of inputs and outputs of the steps are further process-related criteria that will be regarded in the context of these steps.

1. Identify system goals	12. Define agent behavioural interface (e.g., capabilities, services, contracts, ...)
2. Identify system tasks/behaviour	13. Specify system architecture (i.e., overview of all components and their connections)
3. Specify use case scenarios	14. Specify organisational structure/control regime/inter-agent social relationships
4. Identify roles	15. Model MAS environment (e.g., resources, facilities, characteristics)
5. Identify agent classes	16. Specify agent-environment interaction mechanism
6. Model domain conceptualisation	17. Specify agent inheritance and aggregation
7. Specify acquaintances between agent classes	18. Instantiate agent classes
8. Define interaction protocols	19. Specify agent instances deployment
9. Define content of exchanged messages	
10. Specify agent architecture	
11. Define agent mental attitudes (e.g., goals, beliefs, plans, commitments, ...)	

Table 14.4: 19 required steps for an AOSE process according to Tran et al. (2005)

In addition, the technique-related criteria listed in Table 14.5 are evaluated for each of the steps. While they, again, introduce subjective measures, it is important to determine whether single steps are merely mentioned or described extensively and supported by examples. The criteria will be used in a metric to evaluate how well the steps are covered by a process.

Criterion	Description
Availability of techniques and heuristics	Which techniques are used in each step and how are the models produced through them?
Ease of understanding and usability of the techniques	Are the techniques described in a way that is easy to understand and to apply for a developer?
Provision of examples	Does the process description detail the techniques with examples?

Table 14.5: Technique-related criteria for the evaluation of singular steps within agent-oriented software engineering processes according to Tran et al. (2005)

Criteria evaluation and justification In evaluating these criteria, it is necessary to justify the answers. The criterion "Size of MAS" can be answered by a simple number, but the answer has to be based on characteristics of the process which should be made explicit. If a process provides guidance

for a certain system size, this guidance and how it supports the developer should be mentioned. The mere resistance of such a guidance, however, does not ensure that it is usable: it has to be effective and helpful to the development team as well. Unfortunately, the original comparison papers do not contain such justifications. However, we will provide the rationales for our assessment of PosoMAS in Section 14.4.

14.2 Emergency Response Systems

The Self-Organising Emergency Response System (So-ERS), introduced in Section 2.9, is a case study that is at the same time highly mission-critical as well as inherently distributed. The goal of the system is to allow robust communication between first responders and command staff during emergencies. The software engineering process has been applied to this scenario. Selected results of this application will be shown in the following sorted by the phase in which they were created, after briefly outlining the characteristics of the case study and the applicability of PosoMAS.

Characteristics of the So-ERS and Applicability of PosoMAS

The So-ERS constitutes a system of systems as defined in Section 3.2. The individual elements of the system can operate independently, individual teams are managed independently, the first responders and the command staff are geographically distributed, the system leverages synergies to achieve emergent behaviour in catastrophes, and changes to the deployed system are gradual and evolutionary. The system is organised hierarchically as defined by the command structures of police, fire department, and other emergency responders. Parts of the hierarchy, however, can be changed in a human-controlled self-organisation process to assemble teams of first responders that are most suited for the individual incidents. Data collected in the field is assessed for its credibility and thus for its potential to contribute to decisions made on all levels of the hierarchy. Data is aggregated when propagated up the hierarchy and decisions and commands are refined when propagated down the hierarchy.

PosoMAS is applicable to this case study since it constitutes a large-scale open multi-agent systems with strong interactions with its environment. While self-organisation is not a main concern in the case study, it still allows the adaptation of structures. System structure, however, is a main concern. As the system involves many stakeholders, it is crucial to define standards as well as interoperable protocols and communicate requirements clearly and unambiguously. Further, the So-ERS is mission-critical and thus profits from concise architectural descriptions.

Inception Phase

The inception phase was conducted with three iterations in total. During these iterations, a shift from initial process planning towards requirements analysis could be seen. The first iteration was focused on preparing the project, agreeing on the scope of the project in a vision document, and creating a project plan.

Activity: Initiate Project Two tasks are contained within this activity: *Develop Technical Vision* and *Plan Project*. The vision, printed below contains the scope and the main requirements that the final product has to fulfil. It is a result of the initial business needs of the customer and defines the product that is created in the course of the process. The creation of the vision coincides with a first version of the *Glossary* that contains the important terms and the vocabulary that will be used by the development team and the stakeholders in all communication about the project.

Work Product: Vision

In a Self-Organising Emergency Response System, all personnel is equipped with handheld devices and an array of sensors that capture vital data about the current situation at the scene of the incident and that communicate this data up the chain of command. The purpose of the So-ERS is to create an ad-hoc network between the devices of the team members and propagate data up the chain of command while propagating orders down. Additional data, e.g., from fire protection systems or sensors in the environment also has to incorporated.

The devices of the first responders and relief units at the scene of the emergency are automatically organised into teams based on the structure provided by the team leaders or the command staff at the scene. This structure will be based on the equipment available and the immediate needs that present itself. Sensors and devices located in the equipment of an individual first responder will be connected in a Body Area Network (BAN). The data generated is collected by a main device, aggregated and forwarded to either team members or command staff over a wireless ad-hoc network. Depending on the emergency and on the team structure, each of the stakeholders might require different information.

To collect data that is not generated by devices or sensors carried by a first responder, interfaces to the sensors in the environment are exploited. These can either be local (e.g. in case of fire protection systems in buildings that provide data about the sources of fires, which parts of the buildings are locked, or where sprinkler systems are active) or require a connection to a networked system (e.g. in case of large-scale sensor networks for forest fires that are controlled by the National Park Service or a similar organisation). These coordination routes are reflected in the system and incorporated into the structure at an appropriate level.

Information that is collected at a higher level in the command structure will be further aggregated to form an overview of the situation of the individual teams and of the scene of the emergency as a whole. This ensures that higher ranking staff and incident commanders are not overwhelmed with information but are able to see where problems occur and where additional forces or different equipment is required.

The actual communication network used depends on the level of the hierarchy that is regarded. At the level of the individual first responders, a Body Area Network is used. At the level of teams, a wireless ad-hoc network is used. This network allows communication between the team members and to the team leader. It has to be redundant and robust to be able to cope with changes in the environment that sever individual communication links. The communication between team leaders and command staff at the scene of the incident can be accomplished with more traditional wireless networks. To transmit data to and from the scene to situation centres, 3G/4G networks or specialised emergency response networks can be utilised.

In summary: The main purpose of a Self-Organising Emergency Response System (So-ERS) is to connect all members of an incident response team, from the individual medic or firefighter, through officers-in-charge at the scene up to the incident commander and to provide up-to-date information to all personnel involved in the relief effort while preventing information overload and ensuring that orders are distributed down the chain of command.

Two main tasks can thus be identified:

1. Create and maintain stable network connections between first responders and command staff.
2. Collect, aggregate, and distribute data through the network, taking into account the particular information requirements of the individuals involved.

Once the vision and thus the scope of the project has been defined, the project plan can be created. This document contains information about the project organisation, i.e., the team that is going to work on the project, possibly with assignments to roles or content areas, a timetable, including milestones and iteration lengths, as well as information about how progress of the project will be measured. The project plan can be adapted in the course of the process if changes in the environment, the development team, or the project requirements make this necessary. In addition, the project plan contains sections for deployment and “lessons learned” that will be filled with content at the appropriate points in time in the project.

Work Product: Initial Project Plan

1 Introduction
This project plan covers the So-ERS project and gives an overview of the organisation of the project.

2 Project organization
The projects’ work is divided into different content areas. Each content area is worked on by one team of developers. Thus there is also a team leader responsible for each content area. The work is divided into the following areas:

- Project Management
- Requirements
- Analysis & Design
- Change Management
- Architecture
- Implementation
- Test
- Deployment

3 Project practices and measurements
An adapted version of the Open Unified Process will be used for this project. The adjustments will allow this version of the OpenUP to be used for multi-agent systems such as the So-ERS. As PosoMAS (Process for Open Self-Organising Multi-Agent Systems) is based on OpenUP, the process life-cycle consists of an inception phase, an elaboration phase, a construction phase and a transition phase. PosoMAS is distinguished from the OpenUP because it uses different practices in some places. To find requirements, e.g., the practice "goal-driven requirements elicitation" will be used instead of OpenUP's standard use-case based requirements practice. Additionally there will be some modifications in the elaboration phase when it comes to designing the system itself, the agents, and the environment due to the project's inherently distributed nature and its implementation as a multi-agent system. As shown in Section 4 of this project plan, the project's progress is divided into two inception iterations, three elaboration iterations, four construction iterations and two transition iterations.

To measure the project's success, schedule issues, cost issues and growth issues will be analysed at every milestone during the project. These measurements will comprise main questions as:

- Are the documents in a state that they can be used in the next iteration?
- Are there any delays with the documents?
- Is the schedule on time?
- Are the costs lower or higher than expected?
- Is the system getting more complex than expected? Is it still doable?

4 Project milestones and objectives
The main objectives for the So-ERS project can be found in the "Vision" document.

Phase	It.	Primary objectives	Scheduled start or milestone	Duration Estimate
Inception	I1	Identify first system goals and requirements Tailor process according to project needs	05/27/2013	7 days
	I2	Expand Requirements Identify and mitigate obstacles	06/03/2013	7 days
Elaboration	E1	Refine Requirements Create initial design based on conceptual domain model	06/10/2013	7 days
	E2	Refine Requirements Choose most complex requirements for initial prototype Adapt design based on chosen requirements	06/17/2013	7 days
	E3	Implement first vertical prototype Refine designs for agents and system architecture	06/24/2013	7 days
	E4	Finish vertical prototype Test vertical prototype Adapt design according to lessons learned	07/01/2013	7 days
Construction	C1	Design and implement system architecture	07/08/2013	7 days

	C2	Design and implement agent architecture Design and implement agent interaction Design and implement agent organisation	07/15/2013	7 days
	C3	Integrate components of the system Perform integration tests	07/22/2013	7 days
Transition	T1	Perform final adaptations according to test results Finalize system testing Begin documentation	07/29/2013	7 days
	T2	Finish documentation	08/05/2013	7 days

5 Deployment
N/A.

6 Lessons Learned
N/A.

Activity: Plan and Manage Iteration Iteration planning and management is a recurring task in each of the iterations of the process. The main purpose is to select the work items that have to be performed during the iteration, track the progress during the iteration while dealing with problems, new issues, and problems, and finally assessing the results of the iteration. The main work product to perform this is the *Iteration Plan* which includes the objectives of the iteration and the list of work items that will be addressed in the iteration.

Practices Used
From the EPF practice library:
Iterative Development, *Project Process Tailoring*
From the PosoMAS practice library:
None.

The box below shows the iteration plan for an early inception iteration in which first requirements are elicited and process tailoring is not yet completed. The assignment of work items has been slightly altered to accommodate the fact that only one developer performed all the work. Additionally, it focuses on work products that are created or altered during the iteration. This is a personal preference of the developer and shows how the work products can be adapted. The developer felt that listing the work products and their description would be a good way to familiarise himself with the documents. Similarly, a section with issues to work on as well as a list of the tasks that will be tackled in the iteration have been added. The issues list was used in part as a communication device between the developer and the product owner. The developer captured potential problems he encountered in the issues list for discussion with the product owner in the next iteration.

Work Product: Iteration Plan for Iteration I2

1 Key Milestones

Milestone	Date
Iteration start	06/03/2013
Iteration stop	06/10/2013

2 High-level Objectives

- Identify general risks and how to handle them
- Agree on the structure of the process
- Refine requirements, add more obstacles and ways to mitigate them
- Agree on technical approach

3 Work Items and Deliverables
The following work items and deliverables will be addressed in this iteration:

Name	Description	Priority
Vision	Defines the features and scope of the project	1
Glossary	Collection of the most important terms used in the project	1
Project Plan	Contains information about the project objectives, timetable, and process customisations	1
Iteration Plan	Describes objectives, work items, and artefacts that are tackled in the iteration	1
Risk List	Describes the risks remaining in the project and possible mitigation strategies	2
Work Items List	A list containing the scheduled work items	2
Project Defined Process	The customised software engineering process that will be used to develop this project	1
Tools List	A list of tools used to develop the product	3
Conceptual Domain Model	Captures important domain concepts and their relationship	1
System Goals Document	Describes the main system goals and requirements	1
Architecture Notebook	Contains important notes about the software architecture	3

4 Issues
None.

5 Tasks
- Identify main system goals
- Create initial diagram about the physical object and their relationships in the system
- Refine main system goals to requirements
- Identify obstacles for goals and requirements
- Find solutions to mitigate obstacles as well as possible

As mentioned in the iteration plan, the structure of the process needs to be agreed upon early on in the project. For this purpose, the OpenUP defines the *Prepare Environment* sub-activity that contains content from the practice *Project Process Tailoring*. In essence, the tasks contained in this practice and thus the activity allow the customisation of an existing process to fit the requirements, the environment, and the development team of the project at hand. The practice provides a guideline on how the tailoring should be done. Important decisions include defining the scope of the tailoring effort—i.e., should a new process be created, an existing process be altered, new content be introduced—and how the changes should be documented. If any project-specific method content is required, it should be created at this point. The deployment of the process—i.e., making the documentation of the tailored process available to all stakeholders and communicating the process—is the final step.

If the Eclipse Process Framework Composer is used, a new delivery process can be created from existing method content. In this case, such an effort is not necessary, however, since the changes to the original PosoMAS for this project are minor and mainly consist in omitting tasks specified in some practices. As self-organisation does not play a major role, the task *Specify self-organisation algorithm* from the PosoMAS practice *Agent Organisation Design* is omitted. Since only a prototypical implementation was created in the course of the simulation, the tasks that define transformation rules from the practice *Model-driven Observer Synthesis* were omitted as well. The tasks from the practice *Trust-based Interaction Design* are not part of the tailored process, either. Other, minor changes, such as the adaptations of the work products mentioned above, have been introduced as a convention and not explicitly been documented.

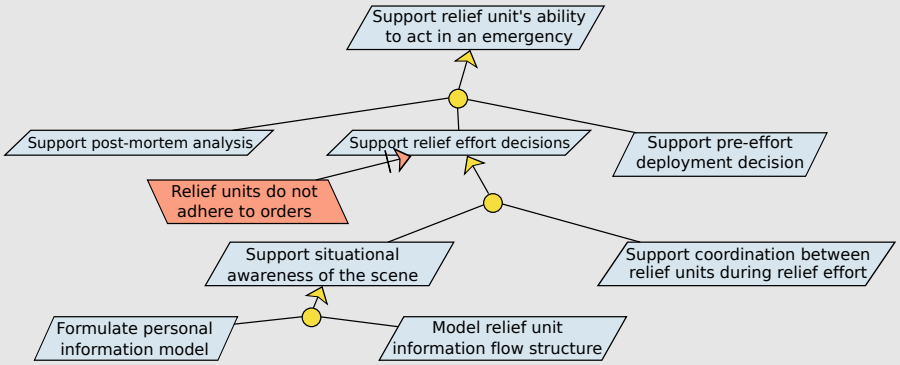
Activity: Identify System Goals and Requirements This activity contains all tasks that deal with requirements elicitation. It is present in all except the last phase but will not receive as much attention in later iterations and phases as in the beginning.

Requirements elicitation in the early stages of the project mainly focuses on high-level system goals that help define the scope of the project and identify the most risky features and those that will create the highest value for the customer. The initial system goals document will be refined in subsequent iterations into more fine-grained requirements, obstacles will be identified and mitigation strategies will be applied. The document is also the basis for selecting work items in the later iterations. According to OpenUP's risk-value life-cycle, the most risky requirements and the ones that create most value for the customer should be tackled first.

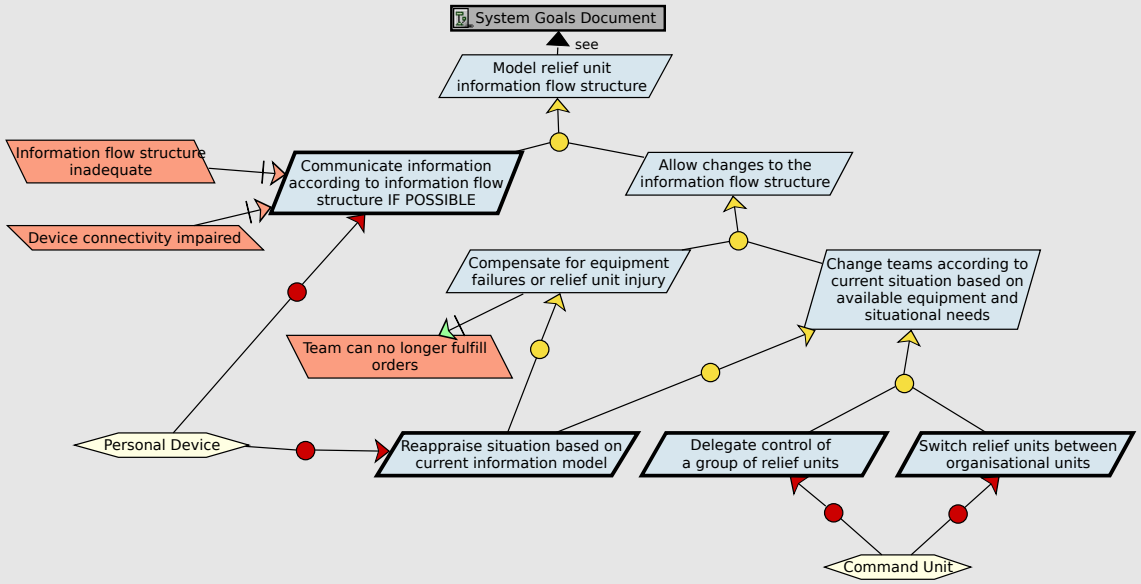
Practices Used
From the EPF practice library:
None
From the PosoMAS practice library:
Goal-driven Requirements Elicitation.

Work Product: System Goals Document in Iteration I3 (Excerpt)

Top-level Goals



Goal: Model relief unit information flow



Elaboration Phase

During elaboration, the design of the product matures. The requirements are realised in the design documents and first runnable builds are created. As the development team's knowledge about the system becomes better, the inherent risk of the project is reduced and the customer value rises.

Activity: Design Architecture Components This activity contains sub-activities and tasks that coordinate the structural design of the solution. The sub-activity *Develop System Architecture* deals with the identification and modelling of the system environment, external interfaces, and supporting infrastructure. The sub-activity *Develop Agent Architecture* is concerned with the internal architecture of the different agent types identified during requirements analysis. All design activities contain specialised tasks that guide the developers in the application of patterns and architectural styles on the designs. In addition, a number of guidelines are available to the development team, e.g., describing methods for *Modular Agent Design* or for the *Separation of System and Agent Levels*.

Practices Used
From the EPF practice library:
Evolutionary Agent Architecture
From the PosoMAS practice library:
Agent System Design, *Evolutionary Agent Design*, *Model-Driven Observer Synthesis*, *Pattern-driven Agent Design*

Since *Model-driven Observer Synthesis* is used for the So-ERS, the task *Define Observation Model* is performed as well, albeit it is omitted in early iterations of the elaboration. Since *Trust-based Interaction Design* does not play a role in the case study, its tasks are omitted. The task *Refine the Architecture*—borrowed from the EPF practice *Evolutionary Architecture*—is included as a reminder that each iteration will see a successive refinement and improvement of the design based on the requirements selected for this iteration.

During the design of the solution, the development team makes a number of important decisions that influence many of the qualitative aspects of the final product. Certain design decisions can, e.g., reduce flexibility with regard to future extensions of the product. In addition, new members of the development team profit from a description of the architecture, its most important elements, and the patterns that have been applied. This information is captured in the *Architecture Notebook*, depicted for the So-ERS below.

Work Product: Architecture Notebook in Iteration E6

1 Purpose
This document describes the philosophy, decisions, constraints, justifications, significant elements, and any other overarching aspects of the system that shape the design and implementation.

2 Architectural Goals and Philosophy
The system itself will be a multi-agent system (MAS) and therefore aspects typical for MAS have to be considered. The system needs to be very robust against changes in its environment but also against changes and failures of its agents, i.e., it needs to function efficiently under unusual conditions because of its use in critical emergency missions. The stability of the network/connection between devices is also very important. Because of this, and the system's highly critical status, the users need to trust in the system. The system is open, meaning it should be possible for agents to enter and leave the system at any time. Scalability is also a very important, but is a factor that is hard to achieve. It will be implemented by the adoption of a hierarchical information flow structure. In this structure, a parallel flow of orders from top to bottom in the hierarchy on the one side will be established while on the other information about the emergency scene, gathered from relief units, will flow from bottom to top within the hierarchy. **3 Assumptions and Dependencies**

- There are some requirements, such as a relief unit not following orders, that can not be addressed by the system. Therefore there are no mitigation plans for such occurrences.
- Whenever the equipment (something gets broken) or the capability of a relief unit changes during the relief effort, the system assumes that those changes are manually made by the relief unit.
- If the connectivity between devices is impaired at any time and therefore no new orders can be forwarded to a relief unit, every relief unit should work on the last received orders until these orders are completed or new ones arrive.

4 Architecturally Significant Requirements

- Requirement: possibility to add new sensor type to the system The architecture gets affected by this requirement, because it is not clear how to manage the possibility to add new sensors to the system. Adding and removing sensors from a unit is handled with dependency injection. So at the start of the system, every ERA will read its configuration file which contains information of the units sensors and how to handle every sensors' data.

5 Decisions, Constraints, and Justifications

- Some requirements in the System Goals Document are assigned to the agent "Device", which is very generic, because of the hierarchy of the units and therefore the hierarchy of the devices, as the agent "Device" is the representative for a unit in real life. It is modelled in this way because of the dynamic allocation of requirements to device/hierarchy level.
- A not yet solved architectural issue is how to bring the location (e.g., of a unit) into the system. Where should this be set? Domain Model? Design Class Diagram? Should it be an attribute or a class?
- The Personal Information Model of a relief unit consists of the following information:
  - `unit`: to identify which units' PIM it is and to access the unitID, team members, superior and subordinates
  - `bodySensorData` : a dictionary of BodySensorModels and the time they were created
  - `externalSensorData` : a dictionary of ExternalSensorModels and the time they were created
  - `superior` : The unitID of the units' superior
  - `subordinates` : a list of the subordinate units
  - `teammembers` : a list of the team member units
  - `task` : the order, the unit has to fulfil
  - `equipment` : the available equipment of the unit
  - `externalPIMS` : a list of the subordinates' Personal Information Models
  - `bodySensorDataRating` : a dictionary of a type in `BodySensorComponents` and a type in Rating; describes/rates the danger of any sensor data
- The environment has different external sensors including the following:
  - `temperature sensor`: measures the temperature at the external sensors location
  - `O2-saturation sensor`: measures the oxygen saturation in the air around the external sensor
  - Each external sensor knows its location
- Every relief unit carries one body sensor, which consists of the following sensor components:
  - temperature sensor component: measures the temperature around the relief unit
  - GPS sensor component: measures the location of the relief unit via GPS
  - O2-saturation sensor component: measures the oxygen saturation in the air around the relief unit
  - gyroscope component: measures the orientation of the relief unit
  - ECG component: measures the activity of the relief units heart
  - ABC component: measures the presence of chemical, biological, or radiological agents in the surrounding air

Agent Architecture Class Diagram

The hierarchy of units is modelled by the classes `Unit`, `UnitModel` and `UnitID` and their relations. In this way, a unit knows all the other units but what differs is the amount of knowledge about each other. Lower located units in the hierarchy only can get the `UnitID` of their superior and therefore it is not possible to get their superiors Personal Information Model (PIM), while a higher located unit can access the PIM of all its subordinates. An order is refined within the system in "sub-orders". Every sub-order also has a "super-order". An order can be issued by a unit and gets communicated to the "executors" by the `CommunicationInterface` of the units' "ERA" (Emergency Response Assistant).

Technical Design

- «component»BodySensor
  - The BodySensor (component, class) itself provides an interface that allows the ERA (agent) to gather data from the sensor
  - The BodySensor consists of different sensor components, such as temperature sensor component etc. (see 3.)
  - Whenever data is retrieved from the body sensor, it creates a new BodySensorModel
  - The BodySensorModel holds all the data from the different body sensor components
  - The SensorComponents class is a generic class of a type of the BodySensorComponents enumeration
- «component»ExternalSensor
  - Every ExternalSensor (component,class) itself provides an interface that allows the ERA (agent) to gather data from it
  - The ExternalSensor consists of different sensor components, such as temperature sensor component etc. (see 3.)
  - Whenever data is retrieved from the external sensor, it creates a new ExternalSensorModel
  - The Sensor class is a generic class of a type of the ExternalSensors
- «agent»ERA
  - The key class of the design. Handles links to the sensor manager, the unit and to the personal information model view.
  - The ERA(class) saves all the BodySensorModels and ExternalSensorModels together with the time in a list
  - The Aggregator(class) creates the PersonalInformationModel(class)
  - The PersonalInformationModel holds various data about a unit (see 5.)
- EraSensorManager: The container for sensors. Has a list of abstract GenericDataInterpreters. Target for dependency injection during bootstrapping.
- GenericDataInterpreter: An interpreter gathers raw sensor data and interprets it to create a model out of it.
- GenericSensor: The sensor itself. Creates data and gives it to the interpreter.
- GenericSensorData: Whenever the sensor gets asked for the actual data, this is created and forwarded to the interpreter.
- GenericDataModel: A model is created out of raw sensor data of the interpreter.
- GenericSensorDataView: The view uses the model to visualize the model and therefore the sensors' data.
- PersonalInformationModelView: This class visualizes the sensor data and the data from the personal information model. It uses the sensor data view of every available sensor to create a composite view of all the sensors views and the information of the PIM.

6 Architectural Mechanisms

Polling: polling will be used to gather sensor data every n seconds.

Publish-Subscribe: publish-subscribe will be used to display sensor data or the personal information model. If a sensor data model changes, the appropriate view gets notified (the same for the PIM)

Model-View-Controller: To display sensor data and personal information models, the model-view-controller pattern is used. There is a view for every type of sensor and for every personal information model. The PersonalInformationModelView then creates a composite view of sensor views and PIMs.

Dependency Injection: The system itself contains only abstract sensor classes. Concrete sensor implementations are injected during bootstrapping, depending on the configuration of a relief unit's gear. This way it is possible to change the dependencies to the sensors at run-time. The sensor.xml file contains the necessary information to bootstrap the system.

7 Layers of the Architectural Framework
Due to the fact that the So-ERS-system is a multi-agent system, we have an arrangement into the following layers:

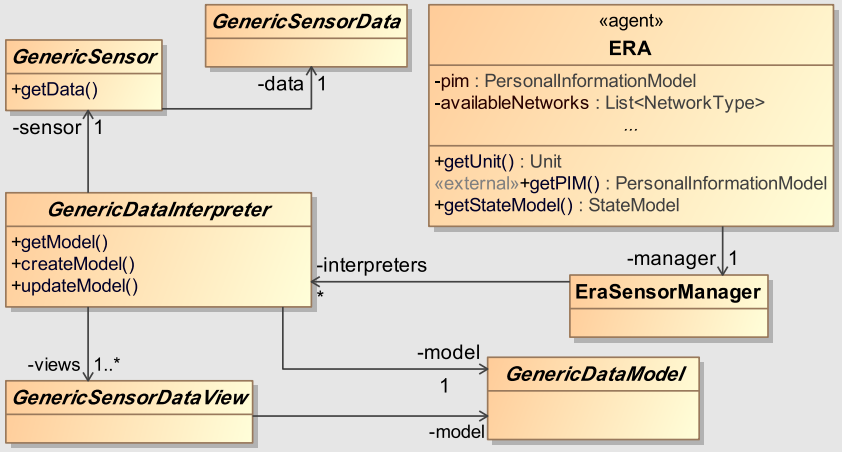
micro-layer: The systems' low-level agents are in this layer. In the case of the So-ERS: the ERA devices every unit carries and the sensors connected to them.

meso-layer: This layer describes the structure of the system. here: the hierarchy of units

macro-layer: This layer describes the systems' behaviour as a whole (corresponding to top-level goals in the System Goals Document).

Apart from the Architecture Notebook, a number of design documents are created. Most of these documents are UML diagrams, depicting different views on the structure defined in the UML models. PosoMAS suggests the creation of models for the architecture of the individual agent types, for the overall system, for the system environment, and for the components interacting in the system. The diagram below is an excerpt from the agent architecture model, depicting the structure of the framework that allows an Emergency Response Agent (ERA) to deal with sensory data. A number of patterns are realised in this design, including Model-View-Controller as outlined in the *Architecture Notebook*. In essence, each ERA uses an `EraSensorManager` to handle sensory input. Raw sensor data is translated into a model which can be represented by one or several views. The `GenericDataInterpreter` serves as the controller for these views. The abstract classes can be implemented by a sensor vendor and instances are created and assigned at runtime using dependency injection.

Work-Product: Agent Architecture (Excerpt) in Iteration E5



Activity: Design System Dynamics The design of the system dynamics consists of three jobs: define the agents' capabilities, define the agents' interactions, and define their organisational structure.

Agent capabilities are building blocks that describe a self-contained functionality. These capabilities are then combined to yield the overall agent behaviour. The main goal of the task and the use of capabilities is to enable reuse of agent functionality among different types of agents and thus reduce implementation overhead. It also serves as a validation of the separation of concerns in the system design, especially with regard to the assignment of functionality to different types of agents.

Work Product: Agent Capabilities List (Excerpt) in Iteration E5

Capability	Requirements
Interaction with ERA device	Allow input of changes in relief status that is not covered by sensors
	Show warning about failed equipment
	Access view for each sensor
	Create view of PIMs
	Create composite view of sensor views and PIM views
Interpret sensor data	Request current sensor data in regular intervals
	Interpret available raw sensor data to generate concrete sensor model
Data aggregation	Integrate hardware failures into personal information model
	Communicate personal information model according to information flow structure IF POSSIBLE
	Combine individual PIMs for situational view
Issue of orders	Communicate order to relief unit IF POSSIBLE
	Assign activities to relief unit with fitting capabilities and proximity to where the activity takes place
	Split order into sequence of more fine-grained orders
	Follow orders UNTIL NEW ONES ARRIVE or all orders are completed
	Reappraise situation based on current information model
Organisational structure changes	Delegate control of a group of relief units
	Switch relief units between organisational units
	Add or remove single units from organisational units

Practices Used
From the EPF practice library:
None.
From the PosoMAS practice library:
Evolutionary Agent Design, *Agent Organisation Design*, *Pattern-driven Agent Design*

The *Design Agent Interactions* task is concerned with the definition of messages exchanged between agents and protocols that define the correct sequence of messages. For these purposes, standard UML models and diagrams can be used again, such as sequence diagrams and state machines. The message content is captured in classes, external interfaces of the agents can be designed in a similar fashion. PosoMAS advocates the use of agent communication languages and provides corresponding guidance. The practice *Pattern-driven MAS Design* includes a task for the pattern-driven design of interaction protocols, mainly based on existing protocols such as the ones defined by FIPA[7].

The design of the organisational structure of the agents is tackled in the sub-activity *Define Agent Organisations*. The organisation present in a system determines the structure in which interactions, information propagation and delegation of control occur in the system. It is the key to scalable and flexible solutions in systems with a large number of agents and the need for cooperation to fulfil the agents' individual goals and the system goals. PosoMAS provides a number of guidances to decide on the best organisation for the problem at hand, including pointers to the scientific literature. The decisions made are captured in a specialised work product slot *[Agent Organisation Design]* that will be used in the later stages of the process to define the interactions required for the self-organisation algorithm.

Activity: Develop Solution Increment The iterative-incremental nature of PosoMAS prescribes the concurrent development of the design and of the system's implementation. A *solution increment* in this sense is a complete build of the product that realises the requirements that have been chosen

[7]The Foundation for Intelligent Physical Agents (FIPA) provides a number of protocol specifications that can be useful when deciding for interaction protocols. These specifications can be found at `http://www.fipa.org/repository/ips.php3`.

for the iteration. In case of the So-ERS, prototypical implementations of a simulated system have been created over the course of several iterations during the elaboration phase. The simulation has been implemented in the multi-agent environment Jadex (Pokahr and Braubach, 2009).

Since Jadex is based on the BDI paradigm, a certain gap between the concepts of the agent architecture and the system dynamics and the concepts required by the multi-agent environment became evident. Most notably, the definition of goals was required, a task not supported by the PosoMAS out of the box[8]. However, the formulation of plans was straightforward by using the definition of the agent functionality and the beliefs follow the agent architecture insofar as classes holding data and their relationships were defined there.

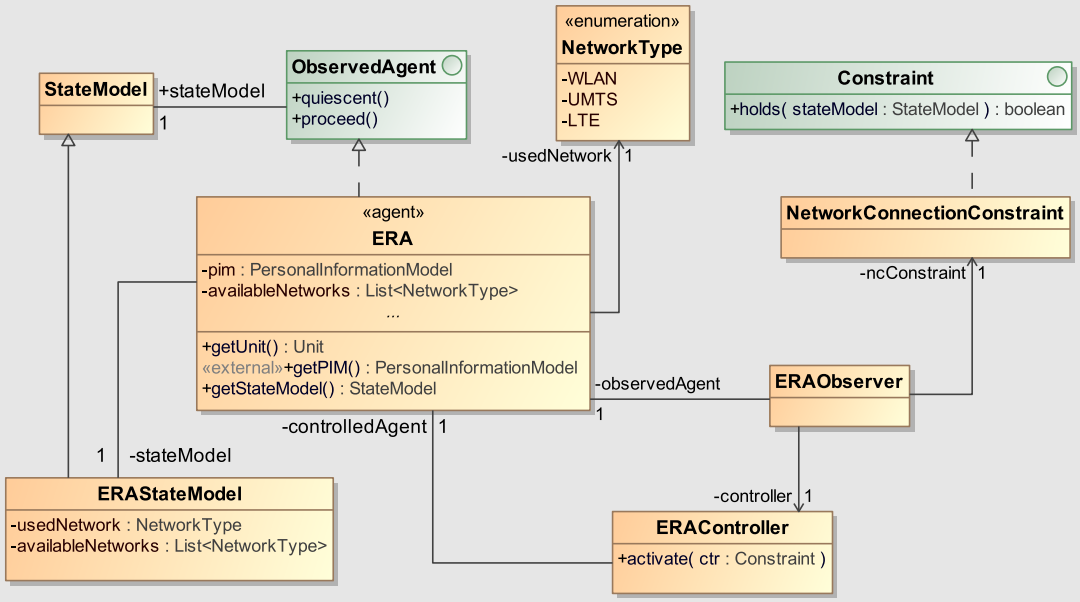
Practices Used
From the EPF practice library:
Test-driven Development, *Continuous Integration*
From the PosoMAS practice library:
Model-driven Observer Synthesis

Construction Phase

While the design activities are reduced in the construction phase according to the assumption that most of the important requirements have already been tackled and dealt with appropriately in the design, it is not yet fixed and feedback from the implementation can still cause changes in the design documents. In general, however, the implementation should be finalised and the preparatory stages to deploy the product should be begun, including creating documentation and training materials.

Activity: Design Architecture Components The design of the architecture components is still refined in these stages. The diagram below shows a number of concepts that have been introduced to deal with the requirement of maintaining a stable network connection even if connectivity is lost in one of the networks available. In addition, the observation model as described in Chapter 6 has to be modelled if the *Model-driven Observer Synthesis* practice is employed. The diagram below shows an excerpt of the agent architecture of an *Emergency Response Agent* (ERA). The ERA is the core element of the So-ERS and bundles all functionality a relief unit has at its disposal in the field. Elements of the generic observation model can be seen in the diagram.

Work Product: Agent Architecture (Excerpt) in Iteration C2



The diagram uses the stereotypes `«agent»` and `«external»`. The former indicates that the class represents an agent and has to be treated accordingly in the implementation. If using a multi-agent

[8]For a discussion of pairing PosoMAS with specialised tasks for BDI design, please refer to Section 13.6.

system framework like, e.g., Jadex, agents are defined in *Agent Definition Files*—XML-files that represent the agent's structure, functionality, and data. A model-to-code transformation has to take this into account. The stereotype thus allows quickly selecting such classes and treating them differently than "regular" data classes. The latter stereotype indicates that an operation is part of the external interface of the agent. It can thus be invoked by calling the method, sending an according message to the agent, or other means depending on the implementation framework. The stereotypes are defined in the *PosoMAS Auxiliary UML Profile*, a reusable asset that is part of the *Evolutionary Agent Architecture* practice.

14.3 Grid-Scale Decentralised Power Management Systems

This section illustrates the application and customisation of PosoMAS to power management systems as introduced in Chapter 2, the steps followed during the process, and shows the most vital artefacts that were generated. Many of the artefacts shown in this thesis are direct of indirect results of the process and many of the descriptions used in the practices, tasks, and guidances of PosoMAS were informed by the experience in creating the power management system.

Since both case studies were conducted in parallel, the iterations used for the power management system are the same as for the So-ERS case study. Therefore, three iterations of the inception phase, seven iterations of the elaboration phase, and two iterations of the construction phase were passed through. Since many parts of the case study were already implemented, the simulated process run focused on requirements engineering and design issues, especially how to reconcile the complex system architecture with the individual agent architectures.

Characteristics of Power Management Systems and Applicability of PosoMAS

As described in Chapter 2, the power management system is a complex, large-scale open self-organising multi-agent system, potentially incorporating thousands of heterogeneous agents in a decentralised system of systems. To achieve the system goals, this heterogeneity and the potential self-interest of the agents has to be reigned in. No central control exists but the system has very strict requirements for internal stability. A number of mechanisms are at play in the system at the same time at different levels of observation.

Since the power management system is fully automated, human users play a secondary role in the case study. However, the large scale of the system requires an elaborated system structure and the dynamics of the system need to be dealt with by a decentralised self-organisation mechanism. Another difference from the So-ERS case study is the presence of different agent types. These can be roughly categorised into agents that represent power plants, agents that represent structural elements such as AVPPs (cf. Chapter 2), and agents that represent other stakeholders such as electricity markets.

PosoMAS is applicable to this case study as it is an example of the system class covered by the process. The system development has a focus on self-organisation and heterogeneity which is covered by the process. Interactions with the environment include a physical infrastructure (the power network including the generators), organisational stakeholders (utilities, power plant operators, legislators), and influences such as the weather and consumers. As in the So-ERS, integrating these diverse stakeholders requires the definition of standards and clear interfaces. As a mission-critical system, the focus on documentation and concise design decisions makes the combination of the PosoMAS practices with the OpenUP a good choice.

Inception Phase

Since the power management case study already had a clearly defined vision as well as initial requirements, the inception phase focussed on creating additional project management artefacts.

Activity: Initiate Project The initial project plan that was created is similar to the one for the So-ERS case study. An existing paper (Steghöfer et al., 2013a) was used as the vision for the project. It contains the relevant information and the desired features of the final product. More interestingly, the initial requirements workshops and the discussions about the vision document lead to an initial risk list, depicted on page 233, that outlines managerial and technical risks the project has to phase in the course of the development. Some of the risks outlined in the document below are very generic

and apply to all or most development efforts—e.g., problems when communicating with management or members of the development team dropping out for various reasons—while others are either specific to the system—e.g., problems with the difference in output between small decentralised power plants and very large ones—or to the system class—e.g., the sheer number of power plants that have to be included. Note that both the descriptions and the mitigation strategies are still somewhat vague at this point. These will be refined during the project. Impact, probabilities, magnitude, etc. can likewise be adapted as the understanding of the system becomes better and new information is revealed. The risk list can also have an impact on the requirements analysis, e.g., by providing obstacles. Likewise, risks identified during the requirements elicitation process can become part of the risk list. It is important, however, to distinguish risks and requirements on the system. While a risk can yield a requirement, it should not be formulated as such.

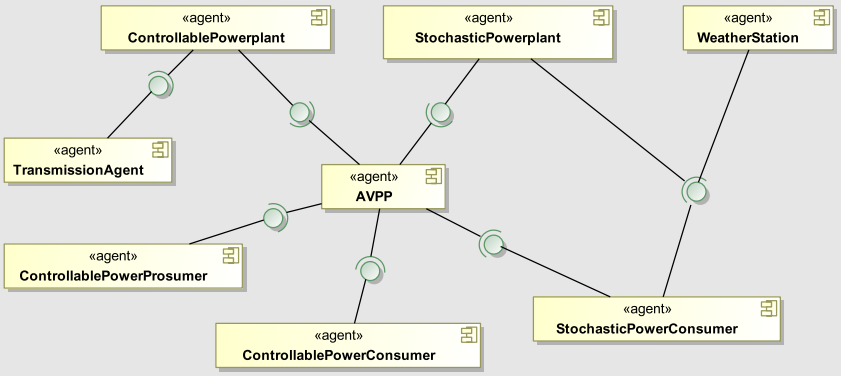
The sub-activity *Project Process Tailoring* yielded only small variations to the prototypical PosoMAS as sketched in Chapter 13. All practices have been used in the tailored process, including *Model-driven Observer Synthesis*, the results of which have been detailed with the corresponding diagrams in Chapter 6 and Chapter 7. In general, the power management case study differs from So-ERS since many decisions have already been made (e.g., selection of an organisational structure and specification of a self-organisation algorithm). The tasks and activities that deal with these decisions are nevertheless simulated to evaluate whether they have the potential to guide the development team towards similar solutions.

Elaboration Phase

The elaboration phase for the power management systems case study included two interesting aspects: the definition of trust-based interactions and the application of a number of agent-based patterns.

Activity: Design Architecture Components The power management case study encompasses a number of different agent types that need to be coordinated and that need to have common interfaces to enable communication between them. To capture these interfaces and the communication structures, a component diagram was used for the system architecture as shown below. The components that are implemented as agents carry the corresponding stereotype `«agent»` defined in the PosoMAS auxiliary UML profile. An interface between two components indicates that information is exchanged between them. A corresponding protocol and message structure has thus to be defined.

Work Product: System Architecture in Iteration E6



However, the first iterations yielded unsatisfactory agent architectures. They were complicated, contained many associations, mixed concerns, and were very difficult to handle. An analysis of the designs and the way they were created revealed that an important aspect of PosoMAS has not been pronounced enough in the preliminary process documentation that was used: the separation of concerns between the different levels of the multi-agent system. In fact, the main problem of the design documents was that agent concerns and system concerns were mixed and that different agent types were defined in the same diagrams. This led to a reformulation of the process description and an extensive refactoring of the design documents. The initial result of the refactoring for the `GenericPowerDevice`, the base agent class for the different types of power plants and consumers is shown below.

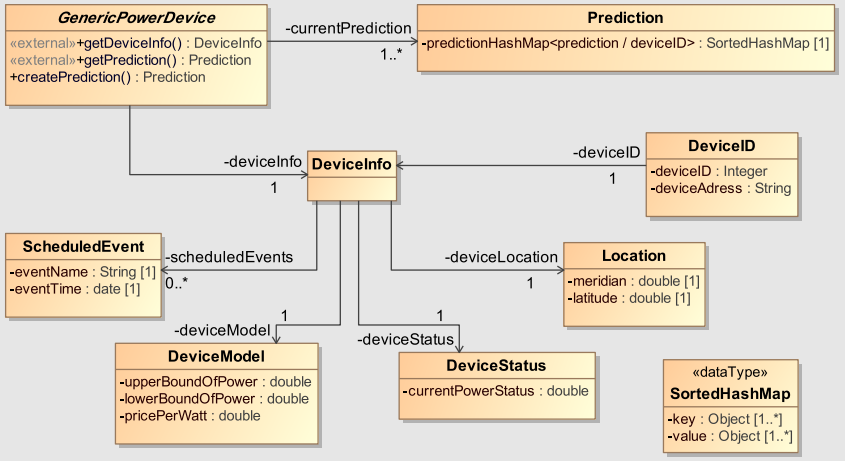
Work Product: Initial Risk List in Iteration I2

Risk ID	Date Identified	Headline	Description	Type	Impact	Probability	Magnitude	Owner	Mitigation Strategy
1	05/07/2013	Opposition by necessarily involved parties	Every lobby and their specific needs have to be addressed in the development of the project to avoid opposition by parties that need to be involved in the project	Man.	3	3	8	Project Manager	Requirements Workshops, Information Sessions, Involvement of all relevant stakeholders at all stages of the project, Clear documentation and communication of all relevant aspects of the project
2	05/07/2013	New technology, Low approval	Normally, every new technology has a low approval at the board of managers	Man.	3	3	6	Project Manager, Product Owner	Keep management informed about progress, outline benefits clearly
3	05/07/2013	Open System	Because of the open system design, many parties participate and everyone of them has their own interests	Man.	3	3	8	Project Manager, Product Owner	See risk No. 1
4	05/07/2013	Faults have huge impact on the grid	Even small faults in the power grid can have huge impact on system stability	Tech.	2	3	5	Analyst, Architect	Identify possible faults and design and implement corresponding response strategies
5	05/07/2013	Complex infrastructure	There is no single point of administration possible/wanted and many small power plants have to be integrated in the overall system	Tech.	2	3	6	Analyst, Architect	Define decentralised control schemes and employ corresponding coordination mechanisms
6	05/14/2013	Unexpected behaviour of agents	Any unexpected behaviour of agents within the system has to be dealt with to ensure system robustness	Tech.	2	2	4	Architect, Development Team	Design and implement appropriate stabilisation mechanisms
7	05/14/2013	Differences between small and big power plants	The difference in production between small and very large power plants can make it difficult to find a suitable system structure and control the system accordingly	Tech.	2	1	3	Architect, Development Team	Define suitable system structures
8	05/10/2013	Unpredicted faults of sub-systems	The system must be able to handle certain failure situations, e.g., the unexpected shutdown of power plants or the failure of transmission lines	Tech.	3	2	6	Analyst, Development Team	Identify failure situations and define mitigation strategies

9	05/10/2013	Production and consumption are asynchronous	It is not possible to predict with 100% accuracy the demand of all users, so there is uncertainty w.r.t. the required schedule	Dom.	2	3	7	Analyst	Measure uncertainty and project expected demand
10	05/12/2013	High fluctuation of participating agents	It is possible that agents (power plants) join or leave the system frequently and thus cause a high volatility in the structures that are established in the system	Dom.	2	2	6	Development Team	
11	05/13/2013	Insufficient data about power grid state	There are currently not enough sensors in the power grid, at the consumers' sites and at the power generators to create a sufficiently detailed state model	Dom.	3	3	8	Product Owner	Equip more sites with sensors, network existing sensors
12	05/07/2013	Important people drop out of the project	Vital members of the development team or key players associated with the project are no longer available	Staff	1	2	7	Project Manager	Communicate within the team, create a common knowledge base, prepare substitution and contingency plans
13	05/07/2013	Too many changes	The product owner requests too many changes during the development of the solution	Man.	1	1	3	Project Manager	Clearly define the requirements during the initial project phases, embrace changes in an agile manner

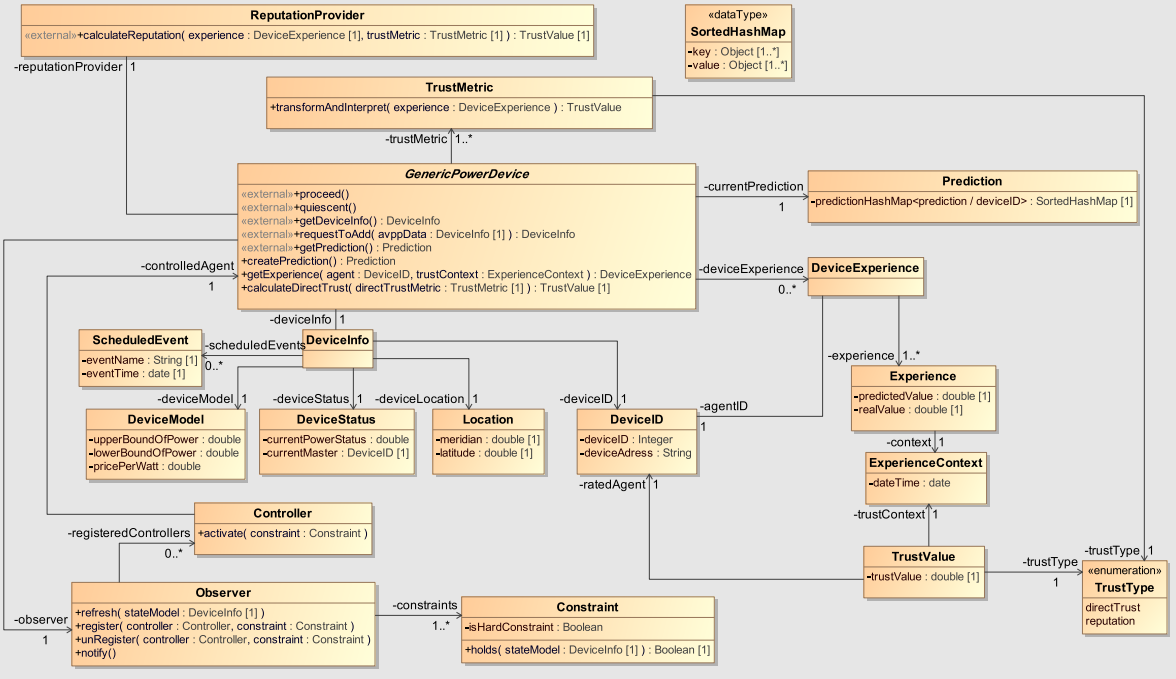
For brevity, *Impact* and *Probability* are defined numerically (1=Low, 2=Medium, 3=High). The *Magnitude* is defined in a more fine-grained way from 1 to 10. These conventions are specific to the project and can be defined by the development team. The type of risk is abbreviated where "Man." stands for "Management", "Tech." for "Technological", and "Dom." for "Domain".

Work Product: Agent Architecture in Iteration E7



After the refactoring, the class is still rather rudimentary and a lot of requirements are not covered yet. This changed in subsequent iterations where a number of patterns were applied and *Trust-based Interaction Design* was used to identify sources of uncertainty and a trust infrastructure was created to deal with these uncertainties. To allow the use of trust values in the system and to exchange trust information with other agents, the reputation pattern from (Anders et al., 2011b) was applied. It defines how experiences from interactions with agents are used to create trust values that allow gauging the benefit of past interactions, or, in this case, the accuracy of predictions provided by other `GenericPowerDevices`. Furthermore, an observer/controller infrastructure was added to the design by applying the corresponding architectural pattern—as introduced in Chapter 3—to allow the system to react to violations of constraints. The resulting architecture for `GenericPowerDevice` is shown below. It retains all of the elements of the version from iteration E7 and integrates them with the additional elements provided by the patterns.

Work Product: Agent Architecture in Iteration E10



Constructon Phase

Since a simulation environment for the power management system already existed, the construction phase was not simulated for this case study. In an actual project, the design mainly drawn up in the elaboration phase is implemented in this phase. The creation of transformation rules for *Model-driven Observer Synthesis* and for model-driven development comes into focus in this phase. As the power management system has to interact intensively with hardware components currently in the field or deployed for the purposes of the system, testing of the integration between the software and the hardware becomes an important activity. Testing and evaluating the self-organisation algorithm (cf. Chapter 4) also comes into focus. Since the system is based on a number of different mechanisms (as witnessed by the contents of this thesis), parametrisation for an eventual role-out should be begun early in the construction phase as well. Different deployment scenarios used as the basis for simulations can be helpful in finding the correct parameter settings.

14.4 Evaluating PosoMAS and Lessons Learned

The simulation runs of PosoMAS showed that the process is accessible for students that perform their first bigger development project with the help of a clearly defined software engineering methodology. The students' feedback was used to improve the process, especially w.r.t. task descriptions and guidances. Therefore, the process evolved slightly while the validation was performed, whereas the overall structure stayed the same. The students also evaluated the process in the context of the evaluation criteria put forward in Section 14.1. The results of this evaluation are detailed below, before discussing the lessons learned from the simulation runs and the creation of the process.

Validation of PosoMAS with Tran et al.'s (2005) Evaluation Criteria

Following Tran et al. (2005), we evaluate PosoMAS for different process criteria, the coverage of different steps and of multi-agent system concepts. A comparison with the processes O-MasE, Prometheus, and ASPECS is provided as well. The data for Prometheus is taken from Tran et al. and (Padgham and Winikoff, 2005), the evaluation for O-MasE is based mainly on (DeLoach and Garcia-Ojeda, 2010), and the one of ASPECS mainly on (Cossentino et al., 2010).

Process-related, Model-related, and Supportive-Feature Criteria The evaluation of the different process criteria for PosoMAS and the comparison with the reference processes is depicted in Table 14.10. Details of the evaluation for PosoMAS and ASPECS are given in Table 14.11. It is important to note that the tables only capture if a process has explicit supportive content for a certain criterion. It is, e.g., very much possible to build proactive multi-agent systems with PosoMAS even though the process does not include specific support for them.

Evaluation Criteria	PosoMAS	O-MasE	Prometheus	ASPECS
Process-Related Criteria				
Development life-cycle	Iterative-incremental risk-value life cycle	Depends on base process	Iterative across all phases	Iterative-incremental life cycle
Coverage of the life-cycle	Conceptualisation, Analysis, Design, (Test, Deployment, Management)	Analysis, Design	Analysis, Design	Analysis, Design, Test, Deployment
Development perspectives	Hybrid	Top-Down	Bottom-Up	Top-Down
Application Domain	Any	Any	Any	Any
Size of MAS	Not Specified	Not Specified	Not Specified	Not Specified
Agent paradigm	Heterogeneous	Heterogeneous	BDI	Holonic
Support for verification	Limited	No[9]	Yes	No
Refinability	Yes	Yes	Yes	Yes

Evaluation Criteria	PosoMAS	O-MasE	Prometheus	ASPECS
Approach towards MAS development	Object-Oriented	Object-Oriented, Role-Based, Goal-Oriented	Object-Oriented, Non-Role-Based	Role-Based, Knowledge Engineering
Meta-model based	No	Yes	No	Yes
Model-Related Criteria				
Syntax and Semantics	High	High	High	High
Model transformations	Yes	Yes	Yes	Yes
Consistency	Yes	Yes	Yes	Yes
Modularity	Yes	Yes	Yes	Yes
Abstraction	Yes	Yes	Yes	Yes
Autonomy	Yes	Yes	Yes	Yes
Adaptability	Yes	Yes	No	Yes
Cooperative behaviour	Yes	Yes	Yes	Yes
Inferential capability	No	Yes	Yes	No
Communication ability	Yes	Yes	Yes	Yes
Personality	No	No	No	No
Reactivity	Yes	Yes	Yes	Yes
Proactivity	No	Yes	Yes	Yes
Temporal continuity	No	No	No	No
Concurrency	No	Yes	No	No
Model Reuse	Yes	Yes	Yes	Yes
Supportive Feature Criteria				
Software and methodological support	Yes	Yes	Yes	Yes
Open systems and scalability	Yes	No	No	Yes
Dynamic structure	Yes	Yes	No	Yes
Performance and robustness	Yes	No	Yes	Yes
Support for conventional objects	Yes	No	Yes	Yes
Support for mobile agents	No	No	No	No
Support for self-interested agents	Yes	No	No	Yes
Support for ontologies	No	No	No	Yes

Table 14.10: Comparison of PosoMAS with O-MasE, Prometheus, and ASPECS based on the evaluation criteria detailed in Section 14.1 and in (Tran et al., 2005).

While **O-MasE** does not explicitly prescribe the BDI agent paradigm, the elements of the used meta-model strongly suggest that the method engineers who created it had current BDI implementations in mind. The OMACS meta-model used defines elements for goals and plans. Desires and Intentions are handled exactly with these concepts in procedural reasoning engines such as Jadex. Indeed, goals exist on two levels in O-MaSE: on the organisational level they direct the creation of an organisational structure to achieve the goals, on the agent level, they define individual goals. The latter goals are achieved by plans (as in BDI), the former goals by assigning roles.

The methodology has a philosophy similar to PosoMAS: method content is defined independently of a process life-cycle. Therefore, some of the aspects—e.g., the development life-cycle—depend on the base methodology that is combined with O-MaSE's method content. PosoMAS, however, is also a process. Therefore, the "development life-cycle" criterion in Table 14.10 is evaluated differently. Unfortunately, the O-MaSE method

[9] O-MaSE claims to have verification abilities, but these are limited to consistency checks between models and do not amount to formal verification of agent behaviours or interaction in the classical sense.

fragments are not available in a repository, although the content is distributed with the download of agentTool, the program also used for the creation of the required models.

Prometheus is the oldest process in the list, but is still one of the best described ones and is still frequently cited. While it does not insist on a meta-model, it prescribes the use of BDI agents. Therefore, the methodology concentrates on the definition of goals, plans, beliefs, and events. The development is mainly from the bottom up since it is very focused on individual agent behaviours and their constitutive elements. One of the bigger differences to the other processes is the fact that adaptability is not explicitly taken into account.

The relatively recent process **ASPECS** is probably the one most similar to PosoMAS. The authors are veterans of agent-oriented software engineering—see, e.g., PASSIM (Cossentino et al., 2008)—and in many regards, ASPECS is the culmination of their collectively learned lessons. The process is also described on a specialised website[10], although the content is very similar to the one from the paper. ASPECS too employs recent AOSE techniques and is aimed at complex systems. However, there are important differences to PosoMAS, especially with regard to the level of abstraction. The main concept ASPECS deals with are holons (cf. Chapter 3). If a system can not be described in holonic terms, the process is not applicable. In addition, ontologies are a central focus of the design of the system. They are used to capture the concepts of the problem domain and used extensively in later stages of the design as well as in the communication between the agents—the traditional use of ontologies in MAS. Cossentino et al. argue that using ontologies enhances the design and allows using AI technologies, possibly semantic-web approaches.

ASPECS also prescribes meta-models for the "agency domain", the "problem domain", and the "solution domain". The first defines concepts such as tasks, groups, roles, and communication, while the second deals with organisations, interactions, and ontologies. The solution domain includes concepts used in the implementation and can contain platform-specific choices. The three meta-models are the basis of the model-driven approach followed by ASPECS in which models are successively refined and transformed from one domain to the next. This ensures consistency of the models but requires the development of transformation rules. PosoMAS and ASPECS are compared according to the criteria from Table 14.10 in more detail in Table 14.11. Please note that (Cossentino et al., 2010) includes a similar comparison with other processes based on a criteria set "inspired" by Tran et al.'s (2005).

Evaluation Criteria	**PosoMAS**	**ASPECS**
Process-Related Criteria		
Development life-cycle	Based on OpenUP's risk-value life cycle.	"Iterative across and within all phases"
Coverage of the life-cycle	Explicit agent-focused support of conceptualisation with goal-based requirements analysis, design with a number of practices, and partially implementation, e.g., with model-driven observer synthesis. Support for conceptualisation and deployment through OpenUP.	Claims to support all phases but is focused on analysis and design, specifically agent and holon design. Covers implementation with rudimentary activities.
Development perspectives	Analysis is performed in a top-down fashion while the design supports both perspectives. Top-down design is used in agent architecture design and system design, bottom-up approaches in the specification of the self-organisation algorithm.	
Application domain	In principle any domain, although adaptivity and heterogeneity are a focus.	In principle any domain, although holonic approach could limit this.
Size of MAS	Aimed at large-scale heterogeneous projects with many agents not under the control of the original developers.	Aimed at large-scale projects that profit from holonic perspective.
Agent paradigm	All method content is formulated in a paradigm-agnostic way.	The agency meta-model does not prescribe a concrete agent paradigm.
Support for verification	The use of constraints within the process allows a limited form of formal analysis.	Although (Cossentino et al., 2010) states that support for formal analysis is available, no information is provided either in the paper or on the website.

[10] `http://www.aspecs.org`, visited October 15, 2013.

Evaluation Criteria	**PosoMAS**	**ASPECS**
Refinability	Advocated by the practices *Evolutionary Agent Architecture* and OpenUP's *Evolutionary Design*.	Stepwise refinement of the models is an important part of the proces.
Approach towards MAS development	While PosoMAS does not explicitly prescribe the approach, its practices are formulated in a way that strongly favours an object-oriented approach.	The definition of roles and of agent organisations is promoted. In addition, ontologies play an important role, so knowledge engineering applies as well.
Meta-model based	No. PosoMAS does not prescribe any meta-models.	ASPECS defines meta-models for the "agency domain", the "problem domain", and the "solution domain"
Model-Related Criteria		
Syntax and Semantics	Syntax and semantics of the models are inherited from UML which is used as a standard modeling language. While this is a different level than using meta-models, it still ensures that the models are based on clearly defined concepts.	Enforced by the meta-models.
Model transformations	Yes, e.g., in *Model-driven Observer Synthesis*.	Yes, the process explicitly promotes using model-driven engineering.
Consistency	Ensured by evolutionary refinement, model transformations, and corresponding guidelines.	Ensured by model transformations and strong adherence to meta-models.
Modularity	Modular agents can be composed from individual components or with other object-oriented techniques.	Modularity based on roles, ontologies, and object-oriented techniques.
Abstraction	Yes, models are possible on different levels of abstraction.	
Autonomy	Yes, by the definition of observation and control models.	Yes, based on individual and collective agent goals.
Adaptability	Yes, by the observation and control models, the self-organisation algorithm, and the trust-based interactions.	Yes, by adapting the holonic structure and with autonomous agent decisions.
Cooperative behaviour	Yes, especially with regard to incorporating untrustworthy agents.	Yes, within holons and between holons.
Inferential capability	Not included.	Not included.
Communication ability	Yes, based on FIPA protocols and *Trust-based Interaction Design*.	Yes, based on FIPA protocols and supported by the ontology.
Personality	Not included.	Not included.
Reactivity	Yes, by the definition of observation and control models.	Yes, with events.
Proactivity	No, there is no specialised method content to account for proactivity.	Yes, based on individual and collective agent goals.
Temporal continuity	Not included.	Not included.
Concurrency	Not included in the processes, but can be modelled with standard UML constructs such as parallel regions in state charts.	
Model Reuse	Yes, models are reused for different purposes within both processes.	
Supportive Feature Criteria		
Software and methodological support	Yes, standard CASE tools can be used. Guidances include many reusable assets.	
Open systems and scalability	Yes, through *Trust-based Interaction Design*, *Agent Organisation Design*, and support material.	Yes, through the use of holons and ontological support.
Dynamic structure	Yes, through *Agent Organisation Design*.	Yes, through the use of holons.

Evaluation Criteria	PosoMAS	ASPECS
Performance and robustness	Yes, through *Trust-based Interaction Design*, *Agent Organisation Design*, and the observation and control models.	Yes, scalability through the use of holons.
Support for conventional objects	Yes, with the use of standard UML.	Yes, with the use of standard UML.
Support for mobile agents	Not included.	Not included.
Support for self-interested agents	Yes, through *Trust-based Interaction Design*	Yes, through individual goals and autonomous decisions about participation in holons.
Support for ontologies	Not included.	Yes, extensively.

Table 14.11: Detailed comparison of PosoMAS and ASPECS according to the evaluation criteria described in Section 14.1.

Process Steps The technique-related criteria shown in Table 14.5 are the basis for a metric to evaluate how well the steps are covered by the different processes. The list below explains the different levels of coverage used in Table 14.12.

None No support is provided (corresponds to "0")

Included The step is present in the methodology but neither a detailed description nor examples are provided (corresponds to "1")

Described The methodology offers a description of the step but no illustrative examples (corresponds to "2A")

Exemplified The methodology provides one or more illustrative examples of the step but does not offer a detailed description (corresponds to "2B")

Full Both a detailed description and an illustrative example are provided (corresponds to "3")

Partial Some aspects of a step are covered (not included in original metric)

None of the processes used in this comparison covers all the process steps. As described in Section 14.1, this is not strictly necessary although the steps capture many important activities within an AOSE process.

Steps	PosoMAS	O-MaSE	Prometheus	ASPECS
Problem Domain Analysis				
1. Identify system goals	Full	Full	None	Full
2. Identify system tasks/behaviour	Full	Full	Full	Full
3. Specify use case scenarios	None	Full	Full	Full
4. Identify roles	None	Full	None	Full
5. Identify agent classes	Full	Full	Full	Full
6. Model domain conceptualisation	Full	Partial	Partial	Partial
Agent Interaction Design				
7. Specify acquaintances between agent classes	Limited	Full	Full	Full
8. Define interaction protocols	Full	Full	Full	Full
9. Define content of exchanged messages	Full	Full	Included	Full
Agent Internal Design				
10. Specify agent architecture	Full	Full	None	Full
11. Define agent mental attitudes (e.g., goals, beliefs, plans, commitments, ...)	Partial	None	Full	Partial
12. Define agent behavioural interface (e.g., capabilities, services, contracts, ...)	Included	None	Full	Partial
System/Environment Design				
13. Specify system architecture (i.e., overview of all components and their connections)	Described	None	None	Partial
14. Specify organisational structure/control regime/inter-agent social relationships	Full	Included	None	Full

Steps	PosoMAS	O-MaSE	Prometheus	ASPECS
15. Model MAS environment (e.g., resources, facilities, characteristics)	Described	None	Described	Partial
16. Specify agent-environment interaction mechanism	Full	None	Full	Partial
17. Specify agent inheritance and aggregation	Included	None	None	Included
Deployment				
18. Instantiate agent classes	None	Full	None	Full
19. Specify agent instances deployment	Included	Full	None	Full

Table 14.12: Coverage of the 19 required process steps introduced in Tran and Low (2005), according to the metric shown below.

ASPECS describes the domain and conceptualises some of its aspects in the problem domain ontology but does not make them available as software engineering concepts.

Coverage of Multi-Agent Concepts Apart from the validation criteria and the process steps, Tran et al. (2005) also provide a list of multi-agent systems concepts and how well the respective processes cover these concepts. The coverage of concepts provided by PosoMAS and the comparison to O-MasE and Prometheus are shown in Table 14.13.

Concepts	PosoMAS	O-MaSE	Prometheus	ASPECS
Problem Domain				
System goals	✓	✓		✓
System roles		✓		✓
System functionality/Tasks		✓	✓	✓
Task responsibilities/Procedures		✓		✓
Design requirements	✓			
Use case/Scenarios		✓	✓	✓
Agent Properties				
Agent classes	✓	✓	✓	✓
Agent instances (including cardinality)	✓		✓	✓
Agent's roles		✓		✓
Agent's functionality	✓	✓		✓
Agent's knowledge/beliefs			✓	
Agent's plans		✓	✓	✓
Agent's goals	✓	✓		✓
Agent's capabilities	✓	✓	✓	✓
Agent mobility				
Agent Interaction				
Interaction pathways	✓	✓	✓	✓
Exchanged messages	✓	✓	✓	✓
Interaction protocols	✓	✓	✓	✓
Interaction constraints		✓		
Conflict resolution mechanisms				
Contracts/commitments	✓			
Ontology				✓
Agent Relationships				
Inheritance	✓			✓
Aggregation	✓			✓
Association	✓			✓
System/Environment				
Co-existing non-agent entities	✓	✓	✓	
Infrastructure/environment facilities	✓	✓		
Organisational Structure	✓	✓		✓
Agent-environment interaction	✓	✓		✓

Concepts	PosoMAS	O-MaSE	Prometheus	ASPECS
Environment characteristics	✓	✓		
Deployment				
Agent architecture		✓		✓
System architecture				✓
Location of agent instances		✓		✓

Table 14.13: Comparison of the support for agent concepts between PosoMAS, MasE, and Prometheus as suggested by Tran et al. (2005). Data for Prometheus from (Tran et al., 2005), data for O-MaSE based on (DeLoach and Garcia-Ojeda, 2010), data for ASPECS based on (Cossentino et al., 2010).

Selecting a Suitable Process

All processes have their characteristics and specific requirements that make them more or less suitable for the development of a certain product. The comparisons in the tables above can provide indications of the strength and weaknesses of the different processes. They are, however, not always easy to interpret, especially since Tran et al. have not necessarily defined their criteria unambiguously. Nevertheless, the list of features and concepts covered by the process, as well as the comparison of the life-cycle, process-related criteria, and supportive-feature criteria can at least give an indication of suitability.

Most processes impose a certain way of thinking about the system under construction on the development team. Prometheus enforces the use of BDI-agents, O-MaSE puts the focus on organisations and away from individual agents, and ASPECS forces the developers to think in terms of ontologies and holarchies. PosoMAS has been designed to be independent of most of these factors but still contains elements that favour certain solutions, e.g., using the Observer/Controller architectural pattern as the basis for adaptation processes in the system. When choosing a process, the development team has to make sure that the perspective taken by the process is compatible with the product. Unfortunately, this is not always clear when the project is initiated. This fact constitutes a classical chicken-and-egg problem: while rather detailed information about the product to develop is necessary to select the most suitable process, this information is only revealed while a process is already selected and under way. Ideally, the definition of the vision and the initial requirements should thus be done before the process is selected and the first iteration is begun.

Personal taste of the development team, e.g., with regard to the agility of the process, plays an important role as well. While some development teams thrive in an agile environment with very little focus on documentation, others are more productive if the documents are prescribed and development steps follow a clear sequential order. AOSE processes tend to be rather heavyweight, possibly due to the fact that their standardisation is far from the level of "traditional" software engineering processes and since agents and their specifics have not yet arrived in mainstream rely on very detailed and extensive documentation to achieve design standards comparable to traditional methodologies. However, as illustrated in Section 13.6, it is possible to embed practices for MAS in agile methodologies and couple the best of both worlds.

Lessons Learned

The development of PosoMAS and the accompanying validation with case studies provided a number of lessons that have been integrated in the process and its documentation. First and foremost, the distinction of architecture areas is vital for the creation of a modular, flexible design. Many of the problems with the initial system design in early iterations were caused by a misunderstandings about which parts of the design were on the agent level, which on the system level and in the environment, and which are part of the organisation design. The clear separation of these areas has thus been described much more thoroughly and according tasks and guidance has been disentangled. This not only leads to a better separation of concern in the resulting design, but in the method content as well.

On a related note, the definition of *scope* has been overhauled during the process. The scope of the system covers what has to be designed by the development team and which parts of the overall system must be accessed through interfaces. In essence, it defines the system boundaries. This has a massive impact on all aspects of the development process: the clearer the scope is defined, the more focused the development team can work. Requirements only need to be defined for those parts of the system that are within the scope of the development effort, designs need only be created for these parts, as well as implementation, unit test, and documentation. However, everything outside the scope the system has to interact with, can not simply be ignored but assumptions must be captured and the environment has to be modelled accordingly. Integration tests have to be designed and performed and the deployment requires integration into the context. While these insights might seem self-evident for the experienced software engineer, the guidance provided by the OpenUP or existing AOSE processes is rather shallow. PosoMAS includes according guidance.

Finally, the level of abstraction differs tremendously between different processes. While the OpenUP is very abstract, without any domain- or problem-specific guidance, Prometheus, ASPECS and other AOSE-processes are very concrete and prescribe solution approaches, techniques, and models in great detail. While the former approach has the advantage to be applicable to arbitrary software projects, the latter excels when a system fits the assumptions the process makes about the product by giving much more hands-on support to the development team. This advantage can, however, be a great disadvantage at the same time since it is rare that a product fits the assumptions perfectly. As outlined before, process tailoring that could reduce this effect is not easily applicable to AOSE processes. PosoMAS tries to find a middle ground between these extremes by providing the development team with guidance without forcing it to adhere to a special paradigm and by formulating method content in a way that lends itself to process customisation and tailoring.

Chapter Summary and Outlook

The results of the evaluation of PosoMAS—the agent-oriented software engineering methodology introduced in the previous chapter—with two different case studies were presented here, including relevant activities and the most important work products that were created during the executions of the process. In addition, a comprehensive catalogue of evaluation criteria is proposed which is then used to characterise PosoMAS and compare it to a number of existing methodologies. The case studies and the comparison show that the process is applicable to the system class and offers competitive features.

Part VI

Conclusion, Discussion, and Outlook

Summary of the findings of the thesis, the evaluation results, discussion, and open research challenges.

Chapter 15

Summary of Research Contributions and Evaluation Results

Summary. The thesis provides contributions in four areas: hierarchical self-organisation, runtime monitoring, constraint optimisation in hierarchical systems, and software engineering for large-scale self-organising multi-agent systems. This chapter summarises these contributions and the evaluation results and thus provides an overview of the achieved outcomes.

The aim of this thesis was to provide techniques to deal with the complexity of large-scale open self-organising multi-agent systems from a structural perspective, from an algorithmic and from a system design point of view. The previous chapters have shown a number of approaches that can help achieve this goal. The following summarises these approaches and the evaluation results that show their applicability to the considered system class. A number of common ideas can be identified that underlie the developed concepts and algorithms: hierarchical system structures, formed by intermediaries; constraint relationships as a way to specify preferences over soft constraints; a focus on models—both from a software engineering and a specification point of view—and their transformation; and embedding the techniques into a software engineering process tailored to large-scale self-organising systems.

Hierarchical Self-Organisation The hierarchical self-organisation algorithm HiSPADA is able to create hierarchical system structures based on the requirements of an application. It uses a set partitioning algorithm for this purpose and relies on the requirements being specified as constraints. These constraints define the need to introduce a new hierarchy layer, to remove an existing one, or to reorganise one that already exists. The hierarchical structure formed by HiSPADA uses the concept of intermediaries to structure the system and allows using a hierarchical observer/controller architecture. Suitable set partitioning algorithms support the creation of structures that correspond to the requirements defined for the hierarchy. A distributed set partitioning algorithm (SPADA) and a jumping frog PSO that uses regionalised knowledge have been developed for use with HiSPADA. The evaluation results show that HiSPADA is able to create hierarchical structures that support the goals of the system. It is also able to react to disturbances in the system and stabilise after such events have occurred.

Runtime Monitoring Based on the novel concept of constraint relationships to express soft constraint and their importance relative to each other as well as based on the restore invariant approach to specify unwanted runtime behaviour with constraints, a framework for behaviour monitoring at runtime has been developed. It is integrated into the software engineering process and uses constraints uncovered during requirements elicitation to derive an observer/controller implementation in a model-driven approach. For this purpose, a variant of the hierarchical observer/controller architecture has been described with software engineering artefacts and combined with application-specific models. The process is highly customisable and exemplified with an extensive case study. Both soft and hard constraints can be monitored, allowing escalating system reactions depending on the violated constraint. The approach is applicable to hierarchical system structures with intermediaries such as the ones created by HiSPADA.

Constraint Optimisation in Hierarchical Systems The adaptation process in self-organising systems can often be expressed as constraint satisfaction or constraint optimisation problems. For use in hierarchical systems, we introduced techniques to combine several individual constraint satisfaction or optimisation models into a composed model that can then be solved by a standard constraint solver. This allows solving hierarchically decomposable tasks whose solution depends on the properties of the individual agents that cooperate to solve

the overall problem. Constraint relationships allow expressing individual preferences that are considered in the solutions of the composed problems. An additional abstraction of synthesised models allows arbitrary numbers of hierarchy levels. The abstraction process uses features of constraint models for hierarchical task allocation problems such as feasible regions. The evaluations of synthesis and abstraction indicate the usefulness of the approaches. It is possible to keep the abstraction error down and at the same time minimise the computational cost of synthesis and abstraction with suitable parameter sets for the problems at hand. In general, a trade-off between abstraction error and abstraction runtime is incurred, especially if many sampling points are used in the abstraction.

Software Engineering for Large-Scale Self-Organising Multi-Agent Systems PosoMAS, the Process for open self-organising Multi-Agent Systems, combines current method engineering techniques with specialised method content for the considered system class and a traditional software engineering life-cycle. It uses practices to describe important engineering concepts for large-scale self-organising systems that allow these techniques to be embedded in other process frameworks. It includes the approaches developed in this thesis, along with a strong distinction between the different levels of observation, the use of patterns and architectural templates, and a focus on trust-based interactions. The applicability of PosoMAS has been evaluated by applying it to two case studies and assessed according to a number of criteria that treat process-related, model-related, technique-related and supportive aspects of AOSE processes. A comparison with other agent-oriented processes based on these criteria has been conducted as well. The customisation of the process for different team sizes and experiences and a combination of the practices with the agile process framework Scrum have been detailed as well.

Chapter 16

Open Research Challenges and Future Directions

Summary. Discusses both future work on the techniques and algorithms presented in this thesis as well as concepts and ideas that are complementary to the research presented here and will play an important role in the advancement of large-scale open self-organising systems.

While this thesis touches many important aspects of large-scale open self-organising systems and provides solutions for a number of problems that arise at design time and at runtime, it necessarily can not cover all relevant issues in such systems. There are a number of possibilities to advance the proposed techniques as outlined in Section 16.1 and there are several foundational aspects that can and will be dealt with in more detail in the future as explicated in Section 16.2.

16.1 Future Directions for Approaches and Techniques Proposed in this Thesis

All of the techniques proposed in this thesis have been developed and evaluated in the context of the power management case study or other large-scale self-organising systems to which they are directly applicable. This strong connection to a realistic system and the ability to implement, test, and validate them in the context of the system, warrant that the contributions serve their purpose in relevant settings and provide a direct benefit to the system. In addition, this strong integration helps identify future directions to improve the algorithms that become apparent in the daily work with the system and its peculiarities. Some of these possible advancements are briefly outlined below for the relevant areas covered here.

Hierarchical Self-organisation and Interfering Feedback Loops

The way SPADA and the Particle Swarm Optimiser—both algorithms have been evaluated for their use as the set partitioning algorithm controlled by HiSPADA—handle changes in the partitioning is not particularly targeted. The leaders in SPADA evaluate their acquaintances in a random order and request the first fitting agent to join their partition. In the PSO, the jumps the particles perform are similarly driven by random exploration and do not necessarily constitute a hill-climbing approach. Future iterations of these algorithms can be enhanced by allowing them to select agents or positions in the solution space that best fit the current requirements of the partition or the system as a whole. Such an adaptation would reqire the algorithms to react differently to the violation of different constraints—if a credibility mix constraint is violated, SPADA would have to select a different agent than if a power source mix constraint is violated. Separate fitness functions for the different optimisation criteria can solve this problem but introduce the necessity to balance those needs, essentially creating a pareto-optimisation problem.

A more pressing concern considering the relatively high solution quality of SPADA and the PSO and the potential complexity of further optimisations is termination of SPADA. As each partition leader evaluates termination locally, it is possible that some partitions remain active in the system and—by requesting agents from terminated partitions to join—re-activate other partitions. This can cause ripple effects in the system where a slight imbalance in one of the partitions leads to an activation of many partitions and essentially prevents global termination. As discussed before, certain parametrisations can help avoid such phenomena but a more effective mechanism would be helpful and allow the use of parameter combinations that give a better solution at the cost of more instability. One possibility to handle this problem is the use of simulated annealing,

i.e., giving each leader a form of "energy" that declines with each action it takes. Once the energy is completely depleted, the partition terminates.

More generally, the way feedback loops and adaptation processes interfere on different hierarchical levels in complex self-organising systems merits an in-depth investigation. Such interference was briefly discussed in Section 8.5 in the context of feedback loops acting on the same degree of freedom. However, relationships between adaptation processes can be more intricate and less obvious. The experience with a complex large-scale self-organising system like the decentralised power management case study shows that small changes in one of the algorithms (e.g., for hierarchical structuration) can have tremendous effects on the outcomes of others (such as scheduling). These interactions are often surprising even for experts who have worked with the system for years and know it in great detail. Designing such complex systems and keeping them stable in operation thus requires new techniques to describe, analyse, and validate the way feedback loops interact with each other.

Model-driven Approach to System Class

The model-driven techniques proposed in this thesis mostly cover the observer part of the observer/controller architecture. While this already simplifies the development of a monitoring infrastructure for large-scale self-organising systems tremendously, the controller part can also be generated at least partially. As outlined in this thesis, the necessary parts in the form of constraints are already available in the requirements. To complement the observer synthesis, it seems natural to develop a kind of controller synthesis that uses the available information about the system structure, the constraints that describe correct information, and a variant of the observer/controller architecture to (semi-)automatically create models for this part of the system and eventually an implementation of the controller.

The techniques developed by Sudeikat et al. (2012) can be helpful in the pursuit of these ideas. They provide the concept of *systemic programming* that makes it possible to combine the top-down development of a system as promoted by most software engineering process including PosoMAS with the bottom-up point of view required for the design of local algorithms in self-organising systems. In essence, the problem is approached from two sides: the individual agent behaviour and inter-agent processes that will eventually lead to globally observable—potentially emergent—behaviour and the overall system behaviour and structure that defines the framework in which the agents operate. Combining these views with the automation techniques developed in this thesis is a viable starting point to further model-driven development for self-organising systems.

Adaptation and Stabilisation with Constraints

Using the constraints acquired during the requirements engineering part of the process for the specification of adaptation and stabilisation in a large-scale self-organising system can be aided by integrating the design of constraint satisfaction and optimisation problems for stabilisation more closely into the engineering process as discussed below. However, even if that is the case, the actual solution algorithms still need to be created manually. As the example of SPADA shows, an actual distributed algorithm that uses local knowledge to solve a problem that can rather easily be specified as a constraint optimisation problem is complex and requires a creative process in which many aspects need to be considered that have nothing to do with the original specification of the problem.

To aid in the creation of these algorithms, it is worthwhile to create a kind of "catalogue" of solution approaches for CSOPs in large-scale open multi-agent systems that do not rely on centralised solvers. These approaches can, e.g., be based on distributed constraint optimisation (Yokoo, 2001) or market-based approaches (see, e.g., Anders et al., 2013b). This would allow applying solution techniques similar to the way patterns are used and alleviate the need for complex algorithm development. Especially in the light of industrial applications in which neither the time nor the expertise for the development of novel algorithmic solutions are available, such a catalogue can potentially help tremendously in fostering the acceptance of agent-oriented techniques.

In the same spirit of reducing the entry threshold, the use of constraint relationships can be simplified by providing appropriate tools for their specification and validation. While it is arguably easier for system designers to designate constraints more important than others instead of ordering them manually, the true semantics of the relationships only become apparent when the designer is presented with different solutions and can reproduce the way the constraints interacted in finding this solution. Therefore, a preference elicitation tool that allows designers to indicate their preferred outcome based on a pool of solutions and that is able to infer the necessary constraint relationships from those stated preferences would allow an interactive construction of complex relationship systems that are intelligible for the designer and offers the opportunity to weigh the different alternatives against each other.

Agent-oriented Software Engineering for Large-scale Self-organising Systems

The consolidation of agent-oriented software engineering methods with the requirements and particularities of the system class considered in this thesis is another concern of future research. On the one hand, providing

more guidelines, examples, and more detailed descriptions will help developers understand the challenges in this system class and the possible solutions available for it. One the other hand, some areas that can be relevant in the development of such systems have not been tackled in great detail yet and additional method content can make the PosoMAS practices and the process applicable to a greater number of concrete systems.

Concretely, areas that deserve more attention are deployment and instantiation, formal methods, and conflict resolution and algorithm design. Testing is another important, yet more complex area that is discussed in the next section. Deployment and instantiation of large-scale self-organising systems are aspects that are mostly ignored by AOSE processes. PosoMAS is no exception so far, even though the issues associated with these topics are very important. The systems of systems paradigm prescribes evolutionary development as the best way to handle complexity with regard to changes in existing systems. We suggest to adopt a similar approach for the initial roll-out of such a system, especially in light of the fact that the system class is especially suited for environments with strong interactions with the physical world and in which existing systems might have to be gradually replaced. Appropriate method content can complement the existing content on deployment from the OpenUP. The integration of formal methods, e.g., for safety analysis or verification into the method content would make these techniques available in the context of the process and assign the proper roles, work products, etc. to the corresponding activities. The integration of conflict resolution and algorithm design would make these issues stand out more clearly in the context of the process. They are already implicitly integrated in practices such as Evolutionary Agent Design and Trust-based Interaction Design but are not yet made explicit. Especially in the context of the future work discussed above and the development of trust-based decision procedures outlined below, these issues merit a more detailed and more prominent place in the process.

Finally, the combination of PosoMAS method content with content from other methodologies should be further developed. This requires both other content to be made available in the form of method fragments or method chunks and appropriate combination techniques being developed. Especially the combination of method content that is based around a different set of work products can be problematic. Suitable abstractions and commonalities have to be defined in such cases and further activities, tasks, or steps to convert work products and to combine different modelling philosophies might become necessary. Nevertheless, such an endeavour is worthwhile since it will make it possible to truly customise processes for specific projects and cherry-pick the best aspects of all method content made available this way.

16.2 Complementary Research Areas and Conceptual Advancement

Complementary aspects to the issues raised in this thesis draw from a broad area of research in self-organising systems, complexity theory, formal modelling and verification, software engineering and social aspects of multi-agent systems. Especially the latter area might be able to provide interesting avenues to deal with challenges occurring in the system class that have not been tackled yet. Since social systems are among the most complex, the concepts that did arise in them might be able to alleviate problems of dealing with uncertainty, with self-interested agents, and with the ever-present threat of fraud and deception. While we know from our own experience that social systems are not able to prevent such phenomena completely, they provide proven mechanisms to deal with them effectively even for very large populations as a kind of social heuristic.

Testing of self-organising multi-agent systems

An important software engineering issue for multi-agent systems that has so far been mostly neglected is the integration of AOSE and MAS techniques with test-driven development and comprehensive offline testing. Even though Nguyen et al. (2011) have proposed first testing techniques for individual agents, it is still unclear how agent ensembles can be tested and how the adaptation processes and flexibility can be covered with automated test suites. In essence, the state explosion problem that is also evident when verifying multi-agent systems (see, e.g., Calinescu et al., 2012) appears in testing as well, exacerbated by the fact that different combinations of agents might behave differently. It is therefore necessary to create environments that allow testing agents and multi-agent systems on different levels of observation and with varying degrees of code and path coverage.

As outlined in (Eberhardinger, 2013), these challenges can at least partially be tackled with techniques put forward in the context of this thesis. The restore invariant approach and its accompanying specification and monitoring approaches might be useful to define test cases and test oracles. They are also the foundation of model-driven techniques which could be extended to generate test cases from the specification of desired behaviour and to identify the necessary levels of observation. PosoMAS offers possibilities to integrate such an approach into a software engineering methodology for large-scale self-organising systems.

Trust-based decision procedures and algorithms

Large-scale open self-organising systems are subject to a number of uncertainties. Open self-organising systems can interact with other systems, the environment, and the user extensively, all of which can influence the system in myriads of unpredictable ways. Additionally, as there is very limited knowledge about and control

over the behaviour of the agents in the system, only very weak assumptions about the system elements can be made, introducing a form of information asymmetry into the systems. These problems become even more dire if systems are safety- or mission-critical, especially in domains such as the aforementioned power management or in emergency response systems, another example of the system class. In such systems, failure can be catastrophic and can cause loss of profit, damage to equipment or the environment, or even injury or death of humans. Therefore, all measures need to be taken to prevent failure, or—if it is inevitable—to deal with failure as graciously as possible.

The uncertainty inherent in the system class adds additional complexity and has to be mitigated both at design time and at runtime. While this thesis mainly deals with aspects that can be tackled at design time, regarding trustworthiness and trust at runtime is a parallel strand of research. The classic notion of trust in the multi-agent systems community is focused on the credibility of agents, i.e., the degree to which they fulfil their commitments, a property that can naturally only be considered at runtime. This view stems mainly from sociological research (Boon and Holmes, 1991) and boils down to the selection of interaction partners in order to maximise the utility of the individual interactions. Economic (Rousseau et al., 1998; Ba and Pavlou, 2002) and computer science literature characterises trust as instrumental to manage *expectations* about others (Mui et al., 2002; Corritore et al., 2003). Often, a strong connection between trust and risk is emphasised (Koller, 1988) as interactions that incur a high risk for the participating agents require a high expectation of the others' willingness to contribute in a beneficial manner. An empirically justified expectation reduces the *uncertainty* about the behaviour of another agent (Ramchurn et al., 2004b). In computing systems, this is captured by a numerical measure, the *trust value* (Marsh, 1994). A trust value is a measure of uncertainty: if an agent's value is very high or very low, the agent is either expected to always behave beneficially or never (Mayer et al., 1995); if the value is between these two extremes, the agent behaves in an unpredictable fashion and thus interactions with it are afflicted with a high uncertainty. This kind of trust is an internal property of the system, one that is established between the interacting agents.

While the algorithms in Chapter 4 and the control models in Chapter 9 introduced ways to incorporate trust in the decisions of an open, self-organising system, many aspects need to be regarded more closely, especially with regard to *trust-based mechanism design* (Dash et al., 2004) in which trust is used as an incentive mechanism. First steps in this direction have been taken (see, e.g., Anders et al., 2013b), but more work on the way trust has to be coupled to incentives, especially on markets, has to be done. The use of a social concept such as trust also raises questions with regard to the usefulness of other social concepts like, e.g., *forgiveness* (Vasalou and Pitt, 2005; Marsh and Briggs, 2009).

Longevity of large-scale open heterogeneous self-organising systems and evolution at runtime

Trustworthiness of a complex system is not a permanent feature but can be fleeting in changing environments and changing circumstances. Power management systems or emergency response systems are not deployed once, run for a short time, and are then quickly superseded by their next generation, but evolve over a long period of time in which they are deployed in the field. During this time, the requirements change as well as the regulatory framework under which they operate, the systems they interact with, and the hard- and software platforms they run on. In addition, the patterns of distribution, the communication network, the users, the currency used, the sub-systems they are composed of, and many other things that were unforeseen at design time change and evolve with them. It is therefore absolutely essential to design systems that are robust to such changes and that have the ability to evolve over time with the environment they operate in.

An important aspect of longevity is the adaptation to a changing legal or regulatory environment and flexibility w.r.t. the business rules imposed by an organisation. All systems in which agents interact are governed by a set of rules. Often, these rules or norms are implicit and not expressed separately. Moses and Tennenholtz (1995) regard the "conventions and restrictions" in place in multi-agent systems as the constituents of a social system that governs the MAS. This notion is reinforced by Mui et al. (2002) who define the agents' adherence to implicit reciprocity norms as the basis of reputation. They describe an agents' environment as an "embedded social network" that is mainly based on an implicit norm of reciprocity of positive actions towards one another.

Normative multi-agent systems (NMAS) make such a system more transparent by explicitly stating behavioural guidelines and more flexible and controllable by allowing to change norms according to environmental circumstance and necessity. Norms do not replace an existing legal framework but can codify laws and statutes as well as augment them with additional organisation-specific guidelines. This constitutes a legislative view on the system where, by default, all behaviours are allowed and the rules constrain the possible behaviours of the agents. Another school of thought understands norms as social laws governing the delegation of institutional power (see, e.g., Artikis et al., 2009). If regarded this way, a norm allows an agent to perform some action that is usually reserved for designated institutions in the system. Additionally, the circumstances must give the agent the power to perform the action (e.g., bids can only be made after a call for proposals has been issued and before the bidding period has ended). This allows to distinguish situations in which the action was *valid*

from those where it was inappropriate. Such a view on norms constitutes a different paradigm. Agents refrain from performing actions they theoretically could perform until they are granted explicit permission to do so.

Such a connotation of norm fits very well to hierarchical market-based systems and conforms to Falcone and Castelfranchi's (2001) notion of trust as the basis of delegation. In the latter paradigm, an implicit norm governs the system that states that agents should not perform an action they have not been explicitly delegated. This potentially reduces the number of norms required in a system as compared to the "everything goes" approach of the former paradigm. However, it can be considered a limitation of agent autonomy, as granting the permission to perform an action is an exception and all agents that are not granted permission are automatically forbidden to perform it.

In any case, the design of norms that will lead to a desired social behaviour is a very hard problem. It has been long known that such a process is NP-complete (Moses and Tennenholtz, 1995). However, this calculation was valid when all possible actions of all the agents that will be part of the system are known. In open, self-organising systems, this assumption does not hold any longer as arbitrary agent implementations can participate in the system in ways that are not foreseeable at design time, introducing emergent behaviour and unpredictable combinations of actions. It is thus necessary to adopt a more liberal approach, much the same way law makers use, and be willing to adapt norms at runtime if they fail to regulate the system sufficiently.

Index

Bibliography

Abdallah, Sherief and Victor Lesser. 'Organization-Based Cooperative Coalition Formation'. *In Int. Conference on Intelligent Agent Technology*, pp. 162–168 (2004).

Abdelaziz, T., M. Elammari, and R. Unland. 'A Framework for the Evaluation of Agent-Oriented Methodologies'. *In Innovations in Information Technology, 2007. IIT '07. 4th International Conference on*, pp. 491 –495 (November 2007).

Abdelaziz, T., M. Elammari, and C. Branki. 'MASD: Towards a Comprehensive Multi-agent System Development Methodology'. *In Proceedings of the OTM Confederated International Workshops and Posters on On the Move to Meaningful Internet Systems*, OTM '08, pp. 108–117. Springer-Verlag, Berlin, Heidelberg (2008).

Abouelela, Mohamed and Mohamed El-Darieby. 'Multidomain hierarchical resource allocation for grid applications'. *In Journal of Electrical and Computer Engineering*, volume 2012 (January 2012).

Al-Hashel, Ebrahim, Bala Balachandran, and Dharmendra Sharma. 'A Comparison of Three Agent-Oriented Software Development Methodologies: ROADMAP, Prometheus, and MaSE'. *In Knowledge-Based Intelligent Information and Engineering Systems*, volume 4694 of *Lecture Notes in Computer Science*, edited by Bruno Apolloni, Robert Howlett, and Lakhmi Jain, pp. 909–916. Springer Berlin / Heidelberg, School of Information Sciences and Engineering, University of Canberra, ACT 2601, Australia (2007).

Allbeck, Jan and Norman Badler. 'Toward representing agent behaviors modified by personality and emotion'. *In Embodied Conversational Agents at AAMAS*, volume 2, pp. 15–19 (2002).

Ambler, Scott W. 'The Agile System Development Life Cycle (SDLC)'. Ambysoft Inc. Website (2012).
URL: *http://www.ambysoft.com/essays/agileLifecycle.html*

Anders, Gerrit, Hella Seebach, Florian Nafz, Jan-Philipp Steghöfer, and Wolfgang Reif. 'Decentralized Reconfiguration for Self-Organizing Resource-Flow Systems Based on Local Knowledge'. *In 8th IEEE Int. Conference and Workshops on Engineering of Autonomic and Autonomous Systems (EASe)*, pp. 20–31 (2011a).

Anders, Gerrit, Jan-Philipp Steghöfer, Florian Siefert, and Wolfgang Reif. 'Patterns to Measure and Utilize Trust in Multi-agent Systems'. *In Self-Adaptive and Self-Organizing Systems Workshops (SASOW), 2011 Fifth IEEE Conference on*, pp. 35 –40. IEEE Computer Society, Washington, D.C., Ann Arbor, MI (Oct. 2011b).

Anders, Gerrit, Christian Hinrichs, Florian Siefert, Pascal Behrmann, Wolfgang Reif, and Michael Sonnenschein. 'On the Influence of Inter-Agent Variation on Multi-Agent Algorithms Solving a Dynamic Task Allocation Problem under Uncertainty'. *In 2012 Sixth IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pp. 29–38. IEEE Computer Society, Washington, D.C. (2012a).

Anders, Gerrit, Florian Siefert, Jan-Philipp Steghöfer, and Wolfgang Reif. 'A Decentralized Multi-agent Algorithm for the Set Partitioning Problem'. *In PRIMA 2012: Principles and Practice of Multi-Agent Systems*, volume 7455 of *Lecture Notes in Computer Science*, edited by Iyad Rahwan, Wayne Wobcke, Sandip Sen, and Toshiharu Sugawara, pp. 107–121. Springer Berlin / Heidelberg (2012b).

———. 'Trust-Based Scenarios – Predicting Future Agent Behavior in Open Self-Organizing Systems'. *In Proc. of the 7th International Workshop on Self-Organizing Systems (IWSOS 2013)*. Springer (May 2013a).

Anders, Gerrit, Jan-Philipp Steghöfer, Florian Siefert, and Wolfgang Reif. 'A Trust- and Cooperation-Based Solution of a Dynamic Resource Allocation Problem'. *In Seventh IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*. IEEE Computer Society, Washington, D.C., Philadelphia, PA (September 2013b).

Andréka, Hajnal, Mark Ryan, and Pierre-Yves Schobbens. 'Operators and Laws for Combining Preference Relations'. *In Journal of Logic and Computation*, volume 12(1), pp. 13–53 (2002).

Ansótegui, Carlos, María L. Bonet, Jordi Levi, and Felip Manyà. 'The Logic Behind Weighted CSP'. *In Trends in Constraint Programming*, pp. 303–316. ISTE (2010).

Appelrath, H.J., H. Kagermann, and C. Mayer, editors. *Future Energy Grid—Migration to the Internet of Energy.* acatech STUDY. acatech – National Academy of Science and Engineering, Munich (2012).

Artikis, Alexander, Marek Sergot, and Jeremy Pitt. 'Specifying norm-governed computational societies'. *In ACM Transactions on Computational Logic (TOCL)*, volume 10(1), p. 1 (2009).

Ba, Sulin and Paul Pavlou. 'Evidence of the effect of trust building technology in electronic markets: price premiums and buyer behavior'. *In MIS Quarterly*, volume 26(3), pp. 243–268 (2002).

Bacchus, Fahiem and Peter van Beek. 'On the conversion between non-binary constraint satisfaction problems'. *In Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, AAAI '98/IAAI '98, pp. 311–318. American Association for Artificial Intelligence, Menlo Park, CA, USA (1998).

Bai, Yu, Jens Brandt, and Klaus Schneider. 'Monitoring distributed reactive systems'. *In High Level Design Validation and Test Workshop (HLDVT), 2012 IEEE International*, pp. 84 –91 (nov. 2012).

Bandyopadhyay, Seema and E.J. Coyle. 'An energy efficient hierarchical clustering algorithm for wireless sensor networks'. *In INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3, pp. 1713–1723 (2003).

Bastos, Ricardo Melo and Marcelo Blois Ribeiro. 'Software Engineering for Multi-Agent Systems III'. *In MASUP: an agent-oriented modeling process for information systems*, edited by Ricardo Choren, Alessandro Garcia, Carlos Lucena, and Alexander Romanovsky, pp. 19–35. Springer-Verlag, Berlin, Heidelberg (2005).

Bauer, Andreas, Martin Leucker, and Christian Schallhart. 'Runtime Verification for LTL and TLTL'. *In ACM Trans. Softw. Eng. Methodol.*, volume 20(4), p. 14 (2011).

Bauer, Bernhard, Jörg P. Müller, and James Odell. 'Agent UML: a formalism for specifying multiagent software systems'. *In First international workshop, AOSE 2000 on Agent-oriented software engineering*, pp. 91–103. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2001).

Bäumler, Simon, Michael Balser, Florian Nafz, Wolfgang Reif, and Gerhard Schellhorn. 'Interactive verification of concurrent systems using symbolic execution'. *In European Journal on Artificial Interlligence (AI Communication)*, volume 23(2-3), pp. 285–307 (2010).

Bäumler, Simon, Gerhard Schellhorn, Bogdan Tofan, and Wolfgang Reif. 'Proving linearizability with temporal logic'. *In Formal Aspects of Computing*, volume 23(1), pp. 91–112 (2011).

Beck, Kent, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, et al. 'The Agile Manifesto' (2001).

Bel, I., A. Valenti, J. Maire, J. M. Corera, and P. Lang. 'Innovative Operation with Aggregated Distributed Generation'. *In Proc. of the 19th International Conference on Electricity Distribution* (2007).

Bernard, Yvonne, Lukas Klejnowski, Christian MÃŒller-Schloer, Jeremy Pitt, and Julia Schaumeier. 'Enduring Institutions and Self-Organising Trust-Adaptive Systems for an Open Grid Computing Infrastructure'. *In Self-Adaptive and Self-Organizing Systems Workshops (SASOW), 2012 IEEE Sixth International Conference on*, pp. 163–168. IEEE (2012).

Bernon, Carole, Marie-Pierre Gleizes, Sylvain Peyruqueou, and Gauthier Picard. 'ADELFE: a methodology for adaptive multi-agent systems engineering'. *In Proceedings of the 3rd international conference on Engineering societies in the agents world III*, ESAW'02, pp. 156–169. Springer-Verlag, Berlin, Heidelberg (2003).

Bernon, Carole, Marie-Pierre Gleizes, Frédéric Migeon, and Giovanna Marzo Serugendo. 'Engineering Self-organising Systems'. *In Self-organising Software*, edited by Giovanna Di Marzo Serugendo, Marie-Pierre Gleizes, and Anthony Karageorgos, Natural Computing Series, pp. 283–312. Springer Berlin Heidelberg (2011).

Beydoun, Ghassan, Graham Low, Brian Henderson-Sellers, Haralambos Mouratidis, Jorge J. Gomez-Sanz, Juan Pavon, and Cesar Gonzalez-Perez. 'FAML: A Generic Metamodel for MAS Development'. *In IEEE Trans. Softw. Eng.*, volume 35(6), pp. 841–863 (November 2009).

Bharosa, Nitesh, JinKyu Lee, and Marijn Janssen. 'Challenges and obstacles in sharing and coordinating information during multi-agency disaster response: Propositions from field exercises'. *In Information Systems Frontiers*, volume 12(1), pp. 49–65 (2010).

Bistarelli, Stefano, Ugo Montanari, and Francesca Rossi. 'Constraint Solving over Semirings'. *In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995*, pp. 624–630 (1995).

———. 'Semiring-based constraint satisfaction and optimization'. *In Journal of the ACM (JACM)*, volume 44(2), pp. 201–236 (1997).

Bistarelli, Stefano, Ugo Montanari, Francesca Rossi, Thomas Schiex, Gérard Verfaillie, and Hélène Fargier. 'Semiring-Based CSPs and Valued CSPs: Frameworks, Properties, and Comparison'. *In Constraints*, volume 4(3), pp. 199–240 (1999).

Boella, Guido and Leendert Torre. 'Organizations in Artificial Social Systems'. *In Coordination, Organizations, Institutions, and Norms in Multi-Agent Systems*, volume 3913 of *Lecture Notes in Computer Science*, edited by Olivier Boissier, Julian Padget, Virginia Dignum, Gabriela Lindemann, Eric Matson, Sascha Ossowski, Jaime Simão Sichman, and Javier Vázquez-Salceda, pp. 198–210. Springer Berlin Heidelberg (2006).

Boella, Guido, Gabriella Pigozzi, and Leonard van der Torre. 'Normative Systems in Computer Science – Ten Guidelines for Normative Multiagent Systems'. *In Normative Multi-Agent Systems*, edited by Guido Boella, Pablo Noriega, Gabriella Pigozzi, and Harko Verhagen, number 09121 in Dagstuhl Seminar Proceedings. Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2009).

Bonabeau, Eric, Guy Theraulaz, and Jean-Louis Deneubourg. 'Mathematical model of self-organizing hierarchies in animal societies'. *In Bulletin of Mathematical Biology*, volume 58(4), pp. 661–717 (1996).

Boon, Susan and John Holmes. 'The Dynamics of Interpersonal Trust: Resolving Uncertainty in the Face of Risk'. *In Cooperation and Prosocial Behaviour*, edited by Robert Hinde and Jo Groebel, pp. 190–211. Cambridge University Press, Cambridge, UK (1991).

Borning, Alan, Robert Duisberg, Bjorn Freeman-Benson, Axel Kramer, and Michael Woolf. 'Constraint hierarchies'. *In Conference proceedings on Object-oriented programming systems, languages and applications*, OOPSLA '87, pp. 48–60. ACM, New York, NY, USA (1987).

Borning, Alan, Bjorn Freeman-Benson, and Molly Wilson. 'Constraint hierarchies'. *In LISP and Symbolic Computation*, volume 5, pp. 223–270 (1992).

Boström, Henrik, Sten F. Andler, Marcus Brohede, Ronnie Johansson, Alexander Karlsson, Joeri van Laere, Lars Niklasson, Marie Nilsson, Anne Persson, and Tom Ziemke. 'On the Definition of Information Fusion as a Field of Research [Elektronisk resurs]'. Technical Report HS- IKI -TR-07-006, Institutionen för kommunikation och information, Skövde (2007).

Bouffard, F. and F.D. Galiana. 'Stochastic security for operations planning with significant wind power generation'. *In Power and Energy Society General Meeting-Conversion and Delivery of Electrical Energy in the 21st Century*, pp. 1–11. IEEE (2008).

Boutilier, Craig, Ronen I. Brafman, Chris Geib, and David Poole. 'A constraint-based approach to preference elicitation and decision making'. *In AAAI Spring Symposium on Qualitative Decision Theory*, pp. 19–28. Citeseer (1997).

Boutilier, Craig, Ronen I. Brafman, Holger H. Hoos, and David Poole. 'Reasoning with conditional ceteris paribus preference statements'. *In Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pp. 71–80 (1999).

Boutilier, Craig, Ronen I. Brafman, Carmel Domshlak, Holger H. Hoos, and David Poole. 'CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements'. *In Journal of Artificial Intelligence Research*, volume 21, pp. 135–191 (2004).

Brafman, R. and C. Domshlak. 'Preference Handling - An Introductory Tutorial'. *In AI Magazine*, volume 30(1), pp. 58–86 (2009).

Brave, Scott, Clifford Nass, and Kevin Hutchinson. 'Computers that care: investigating the effects of orientation of emotion exhibited by an embodied computer agent'. *In International Journal of Human-Computer Studies*, volume 62(2), pp. 161–178 (2005).

Bremer, Jörg, Barbara Rapp, and Michael Sonnenschein. 'Encoding distributed search spaces for virtual power plants'. *In IEEE Symposium on Computational Intelligence Applications In Smart Grid (CIASG)*, pp. 1–8. IEEE, Paris (April 2011).

Brinkkemper, Sjaak, Motoshi Saeki, and Frank Harmsen. 'Meta-modelling based assembly techniques for situational method engineering'. *In Information Systems*, volume 24(3), pp. 209 – 228 (1999).

Brun, Yuriy, Giovanna Marzo Serugendo, Cristina Gacek, Holger Giese, Holger Kienle, Marin Litoiu, Hausi Müller, Mauro Pezzè, and Mary Shaw. 'Engineering Self-Adaptive Systems through Feedback Loops'. *In Software Engineering for Self-Adaptive Systems*, volume 5525 of *Lecture Notes in Computer Science*, edited by Betty H.C. Cheng, Rogério Lemos, Holger Giese, Paola Inverardi, and Jeff Magee, pp. 48–70. Springer Berlin Heidelberg (2009).

Bussmann, Stefan and Klaus Schild. 'Self-organizing manufacturing control: an industrial application of agent technology'. *In MultiAgent Systems, 2000. Proceedings. Fourth International Conference on*, pp. 87–94. IEEE (2000).

Caire, Giovanni, Wim Coulier, Francisco Garijo, Jorge Gomez, Juan Pavon, Francisco Leal, Paulo Chainho, Paul Kearney, Jamie Stark, Richard Evans, and Philippe Massonet. 'Agent Oriented Analysis Using Message/UML'. *In Agent-Oriented Software Engineering II*, volume 2222 of *Lecture Notes in Computer Science*, edited by Michael J. Wooldridge, Gerhard Weiß, and Paolo Ciancarini, pp. 119–135. Springer Berlin Heidelberg (2002).

Calinescu, Radu, Carlo Ghezzi, Marta Kwiatkowska, and Raffaela Mirandola. 'Self-adaptive software needs quantitative verification at runtime'. *In Communications of the ACM*, volume 55(9), pp. 69–77 (September 2012).

Campbell, Adam, Cortney Riggs, and Annie S. Wu. 'On the Impact of Variation on Self-Organizing Systems'. *In 5th IEEE International Conference on Self-Adaptive and Self-Organizing Systems, SASO 2011*, pp. 119–128. IEEE, Ann Arbor, MI, USA (October 2011).

Chalkiadakis, G., V. Robu, R. Kota, Alex Rogers, and Nick Jennings. 'Cooperatives of distributed energy resources for efficient virtual power plants'. *In The 10th International Conference on Autonomous Agents and Multiagent Systems – Volume 2*, AAMAS '11, pp. 787–794. IFAAMAS, Richland, SC (2011).

Chan, Haowen and Adrian Perrig. 'ACE: An Emergent Algorithm for Highly Uniform Cluster Formation'. *In Wireless Sensor Networks*, volume 2920 of *LNCS*, pp. 154–171. Springer (2004).

Cheng, Betty, Pete Sawyer, Nelly Bencomo, and Jon Whittle. 'A Goal-Based Modeling Approach to Develop Requirements of an Adaptive System with Environmental Uncertainty'. *In Model Driven Engineering Languages and Systems*, volume 5795 of *Lecture Notes in Computer Science*, pp. 468–483. Springer Berlin Heidelberg (2009).

Chevaleyre, Yann, Paul E. Dunne, Ulle Endriss, Jérôme Lang, Michel Lemaître, Nicolas Maudet, Julian Padget, Steve Phelps, Juan A. Rodríguez-Aguilar, and Paulo Sousa. 'Issues in multiagent resource allocation'. *In Informatica*, volume 30(1), pp. 3 – 31 (2006).

Choueiry, Berthe Yazid. *Abstraction methods for resource allocation.* Ph.D. thesis, EPFL, Lausanne (1994).

Cockburn, Alistair and Jim Highsmith. 'Agile software development, the people factor'. *In Computer*, volume 34(11), pp. 131–133 (2001).

Colombo, A.W., R. Schoop, and R. Neubert. 'An agent-based intelligent control platform for industrial holonic manufacturing systems'. *In Industrial Electronics, IEEE Transactions on*, volume 53(1), pp. 322–337 (feb. 2005).

Corritore, C.L., B. Kracher, and S. Wiedenbeck. 'On-line trust: concepts, evolving themes, a model'. *In International Journal of Human-Computer Studies*, volume 58(6), pp. 737–758 (2003).

Cossentino, Massimo and Valeria Seidita. 'SPEM Description of ADELFE Process'. Technical report, Consiglio Nazionale delle Ricerche - Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR-CNR), Sede di Palermo Viale delle Scienze edificio 11 90128 Palermo (July 2005). RT-ICAR-PA-05-07.

Cossentino, Massimo, Giancarlo Fortino, Alfredo Garro, Samuele Mascillaro, and Wilma Russo. 'PASSIM – a simulation-based process for the development of multi-agent systems'. *In International Journal of Agent-Oriented Software Engineering*, volume 2(2), pp. 132–170 (Februar 2008).

Cossentino, Massimo, Nicolas Gaud, Vincent Hilaire, Stéphane Galland, and Abderrafiâa Koukam. 'ASPECS: an agent-oriented software process for engineering complex systems'. *In Autonomous Agents and Multi-Agent Systems*, volume 20(2), pp. 260–304 (März 2010).

Cossentino, Massimo, Marie-Pierre Gleizes, Ambra Molesini, and Andrea Omicini. 'Processes engineering and AOSE'. *In Proceedings of the 10th International Conference on Agent-oriented Software Engineering (AOSE'09)*, volume 6038 of *Lecture Notes in Computer Science*, pp. 191–212. Springer-Verlag, Berlin, Heidelberg (2011).

CPLEX. 'IBM ILOG CPLEX Optimizer' (2013).
URL: *http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/*

Dam, Khanh Hoa and Michael Winikoff. 'Comparing Agent-Oriented Methodologies'. *In Proceedings of the Fifth International Bi-Conference Workshop on Agent-Oriented Information Systems*, edited by Paolo Giorgini and Michael Winikoff, pp. 52–59. Melbourne, Australia (2003).

Dardenne, Anne, Axel van Lamsweerde, and Stephen Fickas. 'Goal-directed requirements acquisition'. *In Science of Computer Programming*, volume 20(1-2), pp. 3–50 (1993).

Dash, Rajdeep K, Sarvapali D Ramchurn, and Nicholas R Jennings. 'Trust-based mechanism design'. *In Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pp. 748–755. New York, USA (July 2004).

Dash, R.K., P. Vytelingum, A. Rogers, E. David, and N.R. Jennings. 'Market-based task allocation mechanisms for limited-capacity suppliers'. *In Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, volume 37(3), pp. 391–405 (2007).

Dean, Jeffrey and Sanjay Ghemawat. 'MapReduce: simplified data processing on large clusters'. *In Communications of the ACM*, volume 51(1), pp. 107–113 (2008).

DeLoach, Scott A. and Juan Carlos Garcia-Ojeda. 'O-MaSE – a customisable approach to designing and building complex, adaptive multi-agent systems'. *In International Journal of Agent-Oriented Software Engineering*, volume 4(3), pp. 244–280 (November 2010).

DeLoach, Scott A., Mark F. Wood, and Clint H. Sparkman. 'Multiagent Systems Engineering'. *In International Journal of Software Engineering and Knowledge Engineering*, volume 11(3), pp. 231–258 (2001).

DeLoach, Scott A., Lin Padgham, Anna Perini, Angelo Susi, and John Thangarajah. 'Using three AOSE toolkits to develop a sample design'. *In International Journal of Agent-Oriented Software Engineering*, volume 3(4), pp. 416–476 (May 2009).

Demuth, Birgit. 'The Dresden OCL Toolkit and its Role in Information Systems Development'. *In 13th International Conference on Information Systems Development: Methods and Tools, Theory and Practice Conference, Advances in Theory, Practice and Education (ISD'2004)* (2004).

Demuth, Birgit and Claas Wilke. 'Model and Object Verification by Using Dresden OCL'. *In Proceedings of the Russian-German Workshop Innovation Information Technologies: Theory and Practice*, pp. 81–90 (2009).

Densing, M. 'Hydro-electric power plant dispatch-planning—Multi-stage stochastic programming with time-consistent constraints on risk'. *In Dissertation Abstracts International*, volume 68(04) (2007).

Di Marzo Serugendo, Giovanna, Marie-Pierre Gleizes, and Anthony Karageorgos. 'Self-organization in multi-agent systems'. *In Knowledge Engineering Review*, volume 20(2), pp. 165–189 (June 2005).

Domshlak, Carmel, Francesca Rossi, Kristen Brent Venable, and Toby Walsh. 'Reasoning about soft constraints and conditional preferences: complexity results and approximation techniques'. *In Proceedings of the 18th international joint conference on Artificial intelligence*, pp. 215–220. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2003).

Dorin, Alan and Jon McCormack. 'Self-assembling dynamical hierarchies'. *In Proceedings of the 8th International Conference on Artificial life*, ICAL 2003, pp. 423–428. MIT Press, Cambridge, MA, USA (2003).

Doyle, Jon and Michael McGeachie. 'Exercising qualitative control in autonomous adaptive survivable systems'. *In Self-Adaptive Software: Applications*, volume 2614 of *Lecture Notes in Computer Science*, edited by Robert Laddaga, Howie Shrobe, and Paul Robertson, pp. 158–170. Springer-Verlag, Berlin, Heidelberg (2003).

Dubois, Didier, Hélène Fargier, and Henri Prade. 'The calculus of fuzzy restrictions as a basis for flexible constraint satisfaction'. *In Second IEEE International Conference on Fuzzy Systems*, volume 2, pp. 1131–1136 (1993).

Durfee, Edmund H. 'Practically Coordinating'. *In AI Magazine*, volume 20(1), pp. 99–116 (1999).

Eberhardinger, Benedikt. 'Model-based, Adaptive Testing of Organic Computing Systems'. *In Proceedings of the First Organic Computing Doctoral Dissertation Colloquium (OC-DDC'13)*, volume 2013-06 of *Technical Reports of the Faculty of Applied Computer Science at the University of Augsburg*, edited by Sven Tomforde. Fakultät für Angewandte Informatik der Universität Augsburg (June 2013).

Eberhardinger, Benedikt, Jan-Philipp Steghöfer, Florian Nafz, and Wolfgang Reif. 'Model-driven Synthesis of Monitoring Infrastructure for Reliable Adaptive Multi-Agent Systems'. *In Proceedings of the 24th IEEE International Symposium on Software Reliability Engineering (ISSRE 2013)*. IEEE Computer Society, Washington, D.C., Pasadena, CA (November 2013).

Eclipse Foundation. 'OpenUP' (2013). Accessed September 2nd, 2013.
URL: *http://epf.eclipse.org/wikis/openup/*

Ericsson, G.N. 'Cyber Security and Power System Communication—Essential Parts of a Smart Grid Infrastructure'. *In Power Delivery, IEEE Transactions on*, volume 25(3), pp. 1501–1507 (July 2010).

Esquire Magazine. *The Handbook of Style – A Man's Guide to Looking Good*. Hearst Books (2009).

Eze, Thaddeus, Richard Anthony, Chris Walshaw, and Alan Soper. 'A Methodology for Evaluating Complex Interactions between Multiple Autonomic Managers'. *In The Ninth International Conference on Autonomic and Autonomous Systems (ICAS 2013)*, pp. 53–56. IARIA, Lisbon, Portugal (March 2013).

Fahrioglu, Murat and Fernando L. Alvarado. 'Designing incentive compatible contracts for effective demand management'. *In IEEE Transactions on Power Systems*, volume 15(4), pp. 1255–1260 (2000).

Falcone, Rino and Cristiano Castelfranchi. 'Social trust: a cognitive approach'. *In Trust and deception in virtual societies*, edited by Christiano Castelfranchi and Yao-Hua Tan, pp. 55–90. Kluwer Academic Publishers, Norwell, MA, USA (2001).

Ferber, Jacques, Olivier Gutknecht, and Fabien Michel. 'From Agents to Organizations: An Organizational View of Multi-agent Systems'. *In Agent-Oriented Software Engineering IV*, volume 2935 of *Lecture Notes in Computer Science*, edited by Paolo Giorgini, Jörg P. Müller, and James Odell, pp. 214–230. Springer Berlin Heidelberg (2004).

Fiedrich, Frank and Paul Burghardt. 'Agent-based systems for disaster management'. *In Commun. ACM*, volume 50(3), pp. 41–42 (March 2007).

Fiedrich, Frank, Fritz Gehbauer, and U Rickers. 'Optimized resource allocation for emergency response after earthquake disasters'. *In Safety Science*, volume 35(1), pp. 41–57 (2000).

Firesmith, Donald G and Brian Henderson-Sellers. *The OPEN process framework: An introduction*. Pearson Education (2002).

Fischer, Peter, Florian Nafz, Hella Seebach, and Wolfgang Reif. 'Ensuring correct self-reconfiguration in safety-critical applications by verified result checking'. *In Proceedings of the 2011 Workshop on Organic Computing*, OC '11, pp. 3–12. ACM, New York, NY, USA (2011).

Fox, Mark S. 'Organization structuring: Designing large complex software'. Technical Report Computer Science Technical Report CMU-CS-79-155, Carnegie-Mellon University (December 1979).

Frantz, Frederick K. 'A taxonomy of model abstraction techniques'. *In Proceedings of the 27th conference on Winter simulation*, pp. 1413–1420. IEEE Computer Society (1995).

Freuder, Eugene C. and Richard J. Wallace. 'Partial Constraint Satisfaction'. *In Artificial Intelligence*, volume 58(1-3), pp. 21–70 (1992).

Gamma, Erich, Dirk Riehle, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*. Addison-Wesley, Reading, Mass. (1995).

Garcia, Emilia, Estefania Argente, Adriana Giret, and Vicente Botti. 'Issues for organizational multiagent systems development'. *In Sixth International Workshop From Agent Theory to Agent Implementation (AT2AI-6)*, pp. 59–65 (2008).

García, F. Javier Martínez and José A. Moreno Pérez. 'Jumping Frogs Optimization: a new swarm method for discrete optimization'. Technical Report 3, Documentos de Trabajo del DEIOC, Department of Statistics, O.R. and Computing, University of La Laguna, Tenerife, Spain (2008).

Gärtner, Felix C. 'Fundamentals of fault-tolerant distributed computing in asynchronous environments'. *In ACM Comput. Surv.*, volume 31, pp. 1–26 (March 1999).

Gat, Erann. 'On three-layer architectures'. *In Artificial Intelligence and Mobile Robots*, pp. 195–210 (1998).

Gelain, Mirco, Maria Silvia Pini, Francesca Rossi, and Kristen Brent Venable. 'Dealing with Incomplete Preferences in Soft Constraint Problems'. *In Principles and Practice of Constraint Programming – CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings*, volume 4741 of *Lecture Notes in Computer Science*, edited by Christian Bessière, pp. 286–300. Springer (2007).

Gerkey, B.P. and M.J. Matarić. 'A formal analysis and taxonomy of task allocation in multi-robot systems'. *In International Journal of Robotics Research*, volume 23(9), p. 939 (2004).

Giunchiglia, Fausto and Toby Walsh. 'A theory of abstraction'. *In Artificial Intelligence*, volume 57(2), pp. 323–389 (1992).

Goodloe, Alwyn and Lee Pike. 'Monitoring Distributed Real-Time Systems: A Survey and Future Directions'. Technical Report NASA/CR-2010-216724, NASA Langley Research Center (July 2010).

Görnerup, Olof and James P Crutchfield. 'Hierarchical self-organization in the finitary process soup'. *In Artificial Life*, volume 14(3), pp. 245–254 (2008).

Guesgen, Hans and Joachim Hertzberg. 'Constraint relaxation'. *In A Perspective of Constraint-Based Reasoning*, volume 597 of *Lecture Notes in Computer Science*, pp. 41–56. Springer Berlin / Heidelberg (1992).

Gustafsson, Bjorn. 'OpenUP – The Best of Two Worlds'. *In Methods & Tools* (2008).

Harmsen, Frank and Sjaak Brinkkemper. 'Design and implementation of a method base management system for a situational CASE environment'. *In Proceedings of the Asia Pacific Software Engineering Conference*, pp. 430–438. IEEE (1995).

Heinzelman, Wendi Rabiner, Anantha Chandrakasan, and Hari Balakrishnan. 'Energy-efficient communication protocol for wireless microsensor networks'. *In 2000. Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*, volume 2, p. 10. IEEE (2000).

Henderson-Sellers, B., C. Gonzalez-Perez, and J. Ralyte. 'Comparison of Method Chunks and Method Fragments for Situational Method Engineering'. *In Software Engineering, 2008. ASWEC 2008. 19th Australian Conference on*, pp. 479–488 (2008).

Henderson-Sellers, Brian and Jolita Ralyté. 'Situational Method Engineering: State-of-the-Art Review'. *In Journal of Universal Computer Science*, volume 16(3), pp. 424–478 (feb 2010).

Heo, J.S., K.Y. Lee, and R. Garduno-Ramirez. 'Multiobjective control of power plants using particle swarm optimization techniques'. *In Energy Conversion, IEEE Transactions on*, volume 21(2), pp. 552–561 (2006).

Heylighen, F. et al. 'The science of self-organization and adaptivity'. *In The Encyclopedia of Life Support Systems*, volume 5(3), pp. 253–280 (2001).

Hladik, Pierre-Emmanuel, Hadrien Cambazard, Anne-Marie Déplanche, and Narendra Jussien. 'Solving a real-time allocation problem with constraint programming'. *In Journal of Systems and Software*, volume 81(1), pp. 132–149 (2008).

Hochreiter, R. and G.C. Pflug. 'Financial scenario generation for stochastic multi-stage decision processes as facility location problems'. *In Annals of Operations Research*, volume 152(1), pp. 257–272 (2007).

Horling, Bryan and Victor Lesser. 'A survey of multi-agent organizational paradigms'. *In The Knowledge Engineering Review*, volume 19(04), pp. 281–316 (2004).

IBM Corporation. 'An architectural blueprint for autonomic computing'. *In IBM White Paper* (2006).

Jain, Sanjay and Charles McLean. 'Simulation for emergency response: a framework for modeling and simulation for emergency response'. *In Proceedings of the 35th conference on Winter simulation: driving innovation*, WSC '03, pp. 1068–1076. Winter Simulation Conference (2003).

Jampel, Michael. 'A brief overview of over-constrained systems'. *In Over-Constrained Systems*, volume 1106 of *Lecture Notes in Computer Science*, edited by Michael Jampel, Eugene Freuder, and Michael Maher, pp. 1–22. Springer Berlin / Heidelberg (1996).

Jennex, Murray E. 'Modeling Emergency Response Systems'. *In System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, pp. 22–22 (2007).

Jin, Dongyun, Patrick O'Neil Meredith, Choonghwan Lee, and Grigore Roşu. 'JavaMOP: Efficient Parametric Runtime Monitoring Framework'. *In Proceeding of the 34th International Conference on Software Engineering (ICSE'12)*, pp. 1427–1430. IEEE (2012).

Jones, Clifford B. 'Tentative steps toward a development method for interfering programs'. *In ACM Transactions on Programming Languages and Systems*, volume 5(4), pp. 596–619 (1983).

Jones, Julia C., Mary R. Myerscough, Sonia Graham, and Benjamin P. Oldroyd. 'Honey Bee Nest Thermoregulation: Diversity Promotes Stability'. *In Science*, volume 305(5682) (2004).

Kennedy, J. and R.C. Eberhart. 'A Discrete Binary Version of the Particle Swarm Algorithm'. *In IEEE International Conference on Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation.*, volume 5, pp. 4104–4108 (oct 1997).

Kennedy, James and Russell Eberhart. 'Particle Swarm Optimization'. *In Proc. of the IEEE Int. Conference on Neural Networks*, volume 4, pp. 1942–1948 (1995).

Kent, Stuart. 'Model Driven Engineering'. *In Integrated Formal Methods*, volume 2335 of *Lecture Notes in Computer Science*, edited by Michael Butler, Luigia Petre, and Kaisa Sere, pp. 286–298. Springer Berlin Heidelberg (2002).

Kiefhaber, Rolf, Gerrit Anders, Florian Siefert, Theo Ungerer, and Wolfgang Reif. 'Confidence as a Means to Assess the Accuracy of Trust Values'. *In Proceedings of the 11th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom-2012)*, pp. 690–697. IEEE (2012).

Kienberger, Julian. *Vorgehensmodell für die Entwicklung vertrauenskritischer Multiagentensysteme.* Master's thesis, University of Augsburg, Augsburg, Germany (September 2012).

Kim, Moonjoo, Mahesh Viswanathan, Hanêne Ben-Abdallah, Sampath Kannan, Insup Lee, and Oleg Sokolsky. 'Formally specified monitoring of temporal properties'. *In Proceedings of the 11th Euromicro Conference on Real-Time Systems*, pp. 114–122 (1999).

Kinnebrew, John S. and Gautam Biswas. 'Efficient Allocation of Hierarchically-Decomposable Tasks in a Sensor Web Contract Net'. *In Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology – Volume 02*, WI-IAT '09, pp. 225–232. IEEE Computer Society, Washington, DC, USA (2009).

Kinny, David, Michael Georgeff, and Anand Rao. 'A methodology and modelling technique for systems of BDI agents'. *In Agents Breaking Away*, volume 1038 of *Lecture Notes in Computer Science*, edited by Walter Velde and JohnW. Perram, pp. 56–71. Springer Berlin Heidelberg (1996).

Koestler, Arthur. *The ghost in the machine.* Hutchinson, London (1967).

Kohonen, Teuvo. 'Self-organized formation of topologically correct feature maps'. *In Biological cybernetics*, volume 43(1), pp. 59–69 (1982).

Koller, Michael. 'Risk as a determinant of trust'. *In Basic and Applied Social Psychology*, volume 9(4), pp. 265–276 (1988).

Kotov, Vadim. 'Systems of systems as communicating structures'. *In Object-oriented technology and computing systems re-engineering*, edited by Hussein Zedan and Antonio Cau, pp. 141–154. Horwood Publishing, Ltd., Chichester, USA (1999).

Kroll, Per and Philippe Kruchten. *The Rational Unified Process Made Easy—A Practitioner's Guide to the RUP*. Addison-Wesley Professional (2003).

Kruchten, Philippe. *The Rational Unified Process: an Introduction*. Pearson Education Inc., Boston, MA (2004).

Kyng, Morten, Esben Toftdahl Nielsen, and Margit Kristensen. 'Challenges in designing interactive systems for emergency response'. *In Proceedings of the 6th conference on Designing Interactive systems*, DIS '06, pp. 301–310. ACM, New York, NY, USA (2006).

van Lamsweerde, Axel. 'Goal-oriented requirements engineering: a guided tour'. *In Fifth IEEE International Symposium on Requirements Engineering, 2001. Proceedings.*, pp. 249–262. IEEE (2001).

Lamsweerde, Axel and Emmanuel Letier. 'From Object Orientation to Goal Orientation: A Paradigm Shift for Requirements Engineering'. *In Radical Innovations of Software and Systems Engineering in the Future*, volume 2941 of *Lecture Notes in Computer Science*, pp. 325–340. Springer Berlin Heidelberg (2004).

Larrosa, Javier. 'Node and Arc Consistency in Weighted CSP'. *In AAAI/IAAI*, edited by Rina Dechter and Richard S. Sutton, pp. 48–53. AAAI Press / The MIT Press (2002).

Larrosa, Javier and Thomas Schiex. 'Solving weighted CSP by maintaining arc consistency'. *In Artificial Intelligence*, volume 159(1–2), pp. 1–26 (2004).

Lee, Kangsun and P.A. Fishwick. 'Dynamic model abstraction'. *In Simulation Conference, 1996. Proceedings. Winter*, pp. 764–771 (1996).

Lenaerts, Tom, Dominique Chu, and Richard Watson. 'Dynamical hierarchies'. *In Artificial Life*, volume 11(4), pp. 403–405 (2005).

Leucker, Martin and Christian Schallhart. 'A brief account of runtime verification'. *In The Journal of Logic and Algebraic Programming*, volume 78(5), pp. 293–303 (2009).

Lombardi, P., M. Powalko, and K. Rudion. 'Optimal operation of a virtual power plant'. *In Power Energy Society General Meeting (PES '09)*, pp. 1–6. IEEE (2009).

Low, Graham, Ghassan Beydoun, Brian Henderson-Sellers, and Cesar Gonzalez-Perez. 'Towards Method Engineering for Multi-Agent Systems: A Validation of a Generic MAS Metamodel'. *In Agent Computing and Multi-Agent Systems*, edited by Aditya Ghose, Guido Governatori, and Ramakoti Sadananda, pp. 255–267. Springer-Verlag, Berlin, Heidelberg (2009).

Luo, Yuan, Beth Plale, Zhenhua Guo, Wilfred W. Li, Judy Qiu, and Yiming Sun. 'Hierarchical MapReduce: towards simplified cross-domain data processing'. *In Concurrency and Computation: Practice and Experience* (2012).

Luttrell, Stephen P. 'Hierarchical self-organising networks'. *In First IEE International Conference on Artificial Neural Networks*, pp. 2–6. IEEE (1989).

von Mammen, Sebastian and Jan-Philipp Steghöfer. 'Self-organized middle-out abstraction of complexity'. *In Awareness Magazine* (June 2013). Available online.

———. 'Bring it on, Complexity! Present and future of self-organising middle-out abstraction'. *In The Computer After Me*, edited by Jeremy Pitt, pp. 69–88. Imperial College Press (2014).

Mammen, Sebastian, Jan-Philipp Steghöfer, Jörg Denzinger, and Christian Jacob. 'Self-organized Middle-Out Abstraction'. *In Self-Organizing Systems*, volume 6557 of *Lecture Notes in Computer Science*, edited by Christian Bettstetter and Carlos Gershenson, pp. 26–31. Springer Berlin Heidelberg (2011).

Manna, Zohar and Amir Pnueli. *The temporal logic of reactive and concurrent systems*. Springer-Verlag New York, Inc., New York, NY, USA (1992).

———. *Temporal verification of reactive systems: safety*. Springer-Verlag New York, Inc., New York, NY, USA (1995).

Manthorpe, William H.J. 'The emerging joint system of systems: A systems engineering challenge and opportunity for APL'. *In Johns Hopkins APL Technical Digest*, volume 17(3), p. 305 (1996).

Marsh, Stephen and Pamela Briggs. 'Examining Trust, Forgiveness and Regret as Computational Concepts'. *In Computing with Social Trust*, edited by Jennifer Golbeck, Human-Computer Interaction Series, pp. 9–43. Springer London (2009).

Marsh, Stephen Paul. *Formalising trust as a computational concept*. Ph.D. thesis, University of Stirling (1994).

Marttiin, Pentti and Minna Koskinen. 'Similarities and Differences of Method Engineering and Process Engineering Approaches'. *In Effective Utilization and Management of Emerging Information Technologies*, pp. 420–424 (1998).

Massaguer, Daniel, Vidhya Balasubramanian, Sharad Mehrotra, and Nalini Venkatasubramanian. 'Multi-agent simulation of disaster response'. *In In ATDM Workshop in AAMAS 2006* (2006).

Mayer, Roger C., James H. Davis, and F. David Schoorman. 'An Integrative Model of Organizational Trust'. *In The Academy of Management Review*, volume 20(3), pp. pp. 709–734 (1995).

McDaniel, P. and S. McLaughlin. 'Security and Privacy Challenges in the Smart Grid'. *In Security Privacy, IEEE*, volume 7(3), pp. 75 –77 (2009).

Misra, Jaydev and K. Mani Chandy. 'Proofs of Networks of Processes'. *In Software Engineering, IEEE Transactions on*, volume SE-7(4), pp. 417–426 (July 1981).

Moebius, Nina, Marian Borek, Kurt Stenzel, and Wolfgang Reif. 'SecureMDD: Transformation of a UML application model to a formal specification'. Technical Report 2012-10, Institute for Software & Systems Engineering (2012).

Mookherjee, Dilip. 'Decentralization, hierarchies, and incentives: A mechanism design perspective'. *In Journal of Economic Literature*, volume 44(2), pp. 367–390 (2006).

Moore, Richard, Kelly Reff, James Graham, and Brian Hackerson. 'Scrum at a Fortune 500 Manufacturing Company'. *In Agile Conference (AGILE), 2007*, pp. 175–180. IEEE (2007).

Mösch, Florian, Marek Litza, Adam El Sayed Auf, Erik Maehle, Karl-Erwin Großpietsch, and Werner Brockmann. 'ORCA – Towards an Organic Robotic Control Architecture'. *In Self-Organizing Systems, First International Workshop, IWSOS 2006, and Third International Workshop on New Trends in Network Architectures and Services, EuroNGI 2006, Proceedings*, volume 4124 of *Lecture Notes in Computer Science*, edited by Hermann de Meer and James P. G. Sterbenz, pp. 251–253. Springer, Passau, Germany (2006).

Moses, Yoram and Mosche Tennenholtz. 'Artificial social systems'. *In Computers and Artificial Intelligence*, volume 14, pp. 533–562 (1995).

Motee, N. and B. Sayyar-Rodsari. 'Optimal partitioning in distributed model predictive control'. *In Proceedings of the 2003 American Control Conference*, volume 6, pp. 5300–5305 (June 2003).

Mui, L., M. Mohtashemi, and A. Halberstadt. 'A computational model of trust and reputation'. *In Proc. of the 35th Hawaii International Conference on System Sciences (HICSS'02)*, pp. 188–196. IEEE Computer Society Press (2002).

Nafz, Florian, Frank Ortmeier, Hella Seebach, Jan-Philipp Steghöfer, and Wolfgang Reif. 'A generic software framework for role-based Organic Computing systems'. *In 2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2009, May 18-19, 2009*, pp. 96–105. IEEE, Vancouver, BC, Canada (2009a).

———. 'A Universal Self-Organization Mechanism for Role-Based Organic Computing Systems'. *In Autonomic and Trusted Computing, 6th International Conference, ATC 2009, July 7-9, 2009, Proceedings*, volume 5586 of *Lecture Notes in Computer Science*, edited by Wolfgang Reif, Guojun Wang, and Jadwiga Indulska, pp. 17–31. Springer, Brisbane, Australia (2009b).

Nafz, Florian, Hella Seebach, Jan-Philipp Steghöfer, Simon Bäumler, and Wolfgang Reif. 'A Formal Framework for Compositional Verification of Organic Computing Systems'. *In Autonomic and Trusted Computing – 7th International Conference, ATC 2010, October 26-29, 2010. Proceedings*, volume 6407 of *Lecture Notes in Computer Science*, edited by Bing Xie, Jürgen Branke, Seyed Masoud Sadjadi, Daqing Zhang, and Xingshe Zhou, pp. 17–31. Springer Berlin / Heidelberg, Xi'an, China (2010).

Nafz, Florian, Hella Seebach, Jan-Philipp Steghöfer, Gerrit Anders, and Wolfgang Reif. 'Constraining Self-organisation Through Corridors of Correct Behaviour: The Restore Invariant Approach'. *In Organic Computing – A Paradigm Shift for Complex Systems*, volume 1 of *Autonomic Systems*, edited by Christian Müller-Schloer, Hartmut Schmeck, and Theo Ungerer, pp. 79–93. Springer Basel (2011).

Nafz, Florian, Jan-Philipp Steghöfer, Hella Seebach, and Wolfgang Reif. 'Formal Modeling and Verification of Self-Organizing Systems Based on Observer/Controller-Architectures'. *In Assurances for Self-Adaptive Systems*, volume 7740 of *Lecture Notes in Computer Science*, edited by J. Cámara, R. de Lemos, C. Ghezzi, and A. Lopes, pp. 80–111. Springer Berlin / Heidelberg (2013).

Nah, Fiona Fui-Hoon, Janet Lee-Shang Lau, and Jinghua Kuang. 'Critical factors for successful implementation of enterprise systems'. *In Business Process Management Journal*, volume 7(3), pp. 285–296 (2001).

Ng, Faria and Silvia Sussmann. 'A Personal Travel Assistant for Holiday Selection – A Learning Interface Agent Approach'. *In Information and Communication Technologies in Tourism*, edited by Stefan Klein, Beat Schmid, A.Min Tjoa, and Hannes Werthner, pp. 1–10. Springer Vienna (1996).

Nguyen, Cu D., Anna Perini, Carole Bernon, Juan Pavón, and John Thangarajah. 'Testing in Multi-Agent Systems'. *In Agent-Oriented Software Engineering X*, volume 6038 of *Lecture Notes in Computer Science*, edited by Marie-Pierre Gleizes and JorgeJ. Gomez-Sanz, pp. 180–190. Springer Berlin Heidelberg (2011).

Nicolis, G. 'Physics of far-from-equilibrium systems and self-organisation'. *In The new physics*, volume 11, pp. 316–347 (1989).

Object Management Group (OMG). 'Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification' (2008).
URL: *http://www.omg.org/spec/QVT/1.0/PDF*

O'Brien, Paul and Richard Nicol. 'FIPA – Towards a Standard for Software Agents'. *In BT Technology Journal*, volume 16(3), pp. 51–59 (1998).

Ogston, Elth, Benno Overeinder, Maarten Van Steen, and Frances Brazier. 'A Method for Decentralized Clustering in Large Multi-Agent Systems'. *In Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 789–796 (2003).

OMG. *Software & Systems Process Engineering Meta-Model Specification Version 2.0*. Object Management Group (April 2008).
URL: *http://www.omg.org/spec/SPEM/2.0/*

Omicini, Andrea. 'Agent-Oriented Software Engineering'. *In SODA: Societies and Infrastructures in the Analysis and Design of Agent-Based Systems*, edited by Paolo Ciancarini and Michael J. Wooldridge, pp. 185–193. Springer-Verlag, Berlin, Heidelberg (2001).

Padgham, Lin and Michael Winikoff. *Developing Intelligent Agent Systems*. John Wiley & Sons, Ltd (2005).

Pappala, VS and I. Erlich. 'Power System Optimization under Uncertainties: A PSO Approach'. *In Swarm Intelligence Symposium, 2008. SIS 2008. IEEE*, pp. 1–8. IEEE (2008).

Parsons, Simon and Michael Wooldridge. 'Game Theory and Decision Theory in Multi-Agent Systems'. *In Autonomous Agents and Multi-Agent Systems*, volume 5(3), pp. 243–254 (2002).

Pavón, Juan and Jorge Gómez-Sanz. 'Agent oriented software engineering with INGENIAS'. *In Proceedings of the 3rd Central and Eastern European conference on Multi-agent systems*, CEEMAS'03, pp. 394–403. Springer-Verlag, Berlin, Heidelberg (2003).

Pelikan, Martin and David E. Goldberg. 'Hierarchical Problem Solving by the Bayesian Optimization Algorithm'. *In Proceedings of the Genetic and Evolutionary Computation Conference 2000*, pp. 267–274. Morgan Kaufmann (2000).

Pokahr, Alexander and Lars Braubach. 'From a Research to an Industrial-Strength Agent Platform: Jadex V2'. *In Business Services: Konzepte, Technologien, Anwendungen – 9. Internationale Tagung Wirtschaftsinformatik (WI 2009)*, edited by Hans-Georg Fill Hans Robert Hansen, Dimitris Karagiannis, pp. 769–778. Österreichische Computer Gesellschaft (2 2009).

Pournaras, Evangelos, Martijn Warnier, and Francis Brazier. 'Adaptive agent-based self-organization for robust hierarchical topologies'. *In Adaptive and Intelligent Systems, 2009. ICAIS'09. International Conference on*, pp. 69–76. IEEE (2009).

Prigogine, Ilya and Grégoire Nicolis. *Self-Organization in Non-Equilibrium Systems*. John Wiley and Sons, New York (1977).

Prothmann, Holger, Sven Tomforde, Jürgen Branke, Jörg Hähner, Christian Müller-Schloer, and Hartmut Schmeck. 'Organic Traffic Control'. *In Organic Computing – A Paradigm Shift for Complex Systems*, volume 1 of *Autonomic Systems*, edited by Christian Müller-Schloer, Hartmut Schmeck, and Theo Ungerer, pp. 431–446. Springer Basel (2011).

Pudjianto, D., C. Ramsay, and G. Strbac. 'Virtual power plant and system integration of distributed energy resources'. *In Renewable Power Generation, IET*, volume 1(1), pp. 10–16 (March 2007).

Quinton, Sophie and Rolf Ernst. 'Generalized Weakly-Hard Constraints'. *In Leveraging Applications of Formal Methods, Verification and Validation. Applications and Case Studies*, volume 7610 of *Lecture Notes in Computer Science*, edited by Tiziana Margaria and Bernhard Steffen, pp. 96–110. Springer Berlin Heidelberg (2012).

Rahwan, Talal, Sarvapali D. Ramchurn, Nicholas R. Jennings, and Andrea Giovannucci. 'An Anytime Algorithm for Optimal Coalition Structure Generation'. *In Journal of Artificial Intelligence Research*, volume 34, pp. 521–567 (2009).

Ralyté, Jolita and Colette Rolland. 'An approach for method reengineering'. *In Proceedings of the 20 th International Conference on Conceptual Modeling (ER2001)*, volume 2224 of *Lecture Notes in Computer Science*, pp. 471–484. Springer (2001).

Ralyté, Jolita, Rébecca Deneckère, and Colette Rolland. 'Towards a Generic Model for Situational Method Engineering'. *In Proceedings of the 15th International Conference on Advanced Information Systems Engineering (CAiSE'03)*, volume 2681 of *Lecture Notes in Computer Science*, pp. 95–110. Springer-Verlag, Berlin, Heidelberg (2003).

Ramchurn, S., P. Vytelingum, A. Rogers, and N. Jennings. 'Putting the "Smarts" into the Smart Grid: A Grand Challenge for Artificial Intelligence'. *In Communications of the ACM*, volume 55(4), pp. 86–97 (2012).

Ramchurn, SARVAPALI D., NICHOLAS R. Jennings, Carles Sierra, and Lluis Godo. 'Devising a trust model for multi-agent interactions using confidence and reputation'. *In Applied Artificial Intelligence*, volume 18(9-10), pp. 833–852 (2004a).

Ramchurn, S.D., D. Huynh, and N.R. Jennings. 'Trust in multi-agent systems'. *In The Knowledge Engineering Review*, volume 19(01), pp. 1–25 (2004b).

Rao, Anand S. and Michael P. Georgeff. 'BDI Agents: From Theory to Practice'. *In Proceedings of the First International Conference on Multiagent Systems*, edited by Victor R. Lesser and Les Gasser, pp. 312–319. The MIT Press, San Francisco, CA, USA (June 1995).

Rauber, A., D. Merkl, and M. Dittenbach. 'The growing hierarchical self-organizing map: exploratory analysis of high-dimensional data'. *In IEEE Transactions on Neural Networks*, volume 13(6), pp. 1331–1341 (2002).

Reimann, Stefan. 'Homeostasis and Stability'. Technical Report BibBoS Preprint 97-04-773, University of Bielefeld (1996).
URL: *http://www.math.uni-bielefeld.de/~bibos/preprints/bibos97-773.pdf*

RespectIT. 'Objectiver Homepage' (May 2013).
URL: *http://www.objectiver.com/index.php?id=4*

Reyes-Sierra, Margarita and CA Coello Coello. 'Multi-objective particle swarm optimizers: A survey of the state-of-the-art'. *In International Journal of Computational Intelligence Research*, volume 2(3), pp. 287–308 (2006).

Richter, Urban, Moez Mnif, Jürgen Branke, Christian Müller-Schloer, and Hartmut Schmeck. 'Towards a generic observer/controller architecture for Organic Computing'. *In INFORMATIK 2006 – Informatik für Menschen!*, volume P-93 of *Lecture Notes in Informatics (LNI)*, pp. 112–119. Köllen Verlag (2006).

Rosen, Joseph, Eliot Grigg, Jaron Lanier, Susan McGrath, Scott Lillibridge, David Sargent, and C. Everett Koop. 'The future of command and control for disaster response'. *In Engineering in Medicine and Biology Magazine, IEEE*, volume 21(5), pp. 56–68 (2002).

Rosenblueth, Arturo, Norbert Wiener, and Julian Bigelow. 'Behavior, purpose and teleology'. *In Philosophy of Science*, pp. 18–24 (1943).

Rossi, F., P. Meseguer, and T. Schiex. 'Soft Constraints'. *In Handbook of Constraint Programming*, edited by F. Rossi, P. van Beek, and T. Walsh, chapter 9. Elsevier (2006).

Rossi, Francesca. 'Preferences, Constraints, Uncertainty, and Multi-Agent Scenarios'. *In ISAIM* (2008).

Rossi, Francesca, Kristen Brent Venable, and Toby Walsh. 'Preferences in Constraint Satisfaction and Optimization'. *In AI Magazine*, volume 29(4), pp. 58–68 (2008).

Roşu, Grigore. 'CS422 – Formal Methods in System Design: A Monitor Synthesis Algorithm for Past LTL'. Technical report, Department of Computer Science, University of Illinois at Urbana-Champaign, USA (2007).
URL: *http://fsl.cs.uiuc.edu/images/5/5e/CS477-Spring-2007-pLTL.pdf*

Roşu, Grigore, Feng Chen, and Thomas Ball. 'Synthesizing Monitors for Safety Properties – This Time With Calls and Returns –'. *In Workshop on Runtime Verification (RV'08)*, volume 5289 of *Lecture Notes in Computer Science*, pp. 51–68. Springer (2008).

Rougemaille, Sylvain, Jean-Paul Arcangeli, Marie-Pierre Gleizes, and Frédéric Migeon. 'ADELFE Design, AMAS-ML in Action'. *In Engineering Societies in the Agents World IX*, volume 5485 of *Lecture Notes in Computer Science*, edited by Alexander Artikis, Gauthier Picard, and Laurent Vercouter, pp. 105–120. Springer Berlin Heidelberg (2009).

Rousseau, D.M., S.B. Sitkin, R.S. Burt, and C. Camerer. 'Not so different after all: A cross-discipline view of trust'. *In Academy of management review*, volume 23(3), pp. 393–404 (1998).

Sacerdoti, Earl D. 'Planning in a hierarchy of abstraction spaces'. *In Artificial intelligence*, volume 5(2), pp. 115–135 (1974).

Sage, Andrew P. and Christopher D. Cuppan. 'On the systems engineering and management of systems of systems and federations of systems'. *In Information, Knowledge, Systems Management*, volume 2(4), pp. 325–345 (2001).

Sahinidis, Nikolaos V. 'Optimization under uncertainty: state-of-the-art and opportunities'. *In Computers & Chemical Engineering*, volume 28(6-7), pp. 971–983 (2004).

Sanderson, David, Jeremy Pitt, and Dídac Busquets. 'Interactions of Multiple Self-Adaptive Mechanisms in Multi-Agent Systems'. *In Proceedings of the 2013 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT-13)* (November 2013).

Santos, Cipriano, Xiaoyun Zhu, and Harlan Crowder. 'A Mathematical Optimization Approach for Resource Allocation in Large Scale Data Centers'. Technical Report HPL-2002-64, HP Labs (March 2002).

Schiendorfer, Alexander. *Synthesis and Abstraction of Hierarchically Decomposable Constraint Problems*. Master's thesis, University of Augsburg, Augsburg, Germany (October 2013).

Schiendorfer, Alexander, Jan-Philipp Steghöfer, Alexander Knapp, and Wolfgang Reif. 'Constraint Relationships for Soft Constraints'. *In Proceedings of the 33rd SGAI International Conference on Artificial Intelligence*. Springer (December 2013).

Schiendorfer, Alexander, Jan-Philipp Steghöfer, and Wolfgang Reif. 'Synthesis and Abstraction of Constraint Models for Hierarchical Resource Allocation Problems'. *In Proceedings of the 6th International Conference on Agents and Artificial Intelligence (ICAART)*. SciTePress (March 2014).

Schiex, Thomas, Hélène Fargier, and Gérard Verfaillie. 'Valued Constraint Satisfaction Problems: Hard and Easy Problems'. *In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995*, pp. 631–639 (1995).

Schmeck, Hartmut, Christian Müller-Schloer, Emre Çakar, Moez Mnif, and Urban Richter. 'Adaptivity and self-organization in organic computing systems'. *In ACM Transactions on Autonomous Adaptive Systems*, volume 5, pp. 10:1–10:32 (September 2010).

Schoenharl, Timothy, Greg Madey, Gábor Szabó, and Albert-László Barabási. 'WIPER: A multi-agent system for emergency response'. *In Proceedings of the 3rd International ISCRAM Conference*, pp. 1–7 (2006).

Schulz, C., G. Roder, and M. Kurrat. 'Virtual power plants with combined heat and power micro-units'. *In 2005 International Conference on Future Power Systems* (November 2005).

Schwaber, Ken and Jeff Sutherland. 'The Scrum Guide—The Definitive Guide to Scrum: The Rules of the Game'. *In Scrum.org* (2011).

Schweppe, Fred C. and Sanjoy K. Mitter. 'Hierarchical system theory and electric power systems'. *In Proceedings of the Symposium on Real-time Control of Electric Power Systems*. Elsevier Publishing Company (1972).

Seebach, Hella, Florian Nafz, Jan-Philipp Steghöfer, and Wolfgang Reif. 'A Software Engineering Guideline for Self-Organizing Resource-Flow Systems'. *In IEEE International Conference on Self-Adaptive and Self-Organizing System (SASO)*, pp. 194–203. IEEE Computer Society, Washington, D.C., Budapest, Hungary (2010).

———. 'How to Design and Implement Self-organising Resource-Flow Systems'. *In Organic Computing – A Paradigm Shift for Complex Systems*, edited by Christian Müller-Schloer, Hartmut Schmeck, and Theo Ungerer, Autonomic Systems, pp. 145–161. Springer Basel (2011).

Seidita, Valeria, Massimo Cossentino, and Salvatore Gaglio. 'A repository of fragments for agent systems design'. *In Proceedings Of the Workshop on Objects and Agents (WOA06)*, pp. 130–137. Catania, Italy (September 2006).

Sendall, Shane and Wojtek Kozaczynski. 'Model transformation: the heart and soul of model-driven software development'. *In IEEE Software*, volume 20(5), pp. 42–45 (2003).

Shalizi, Cosma Rohilla, Kristina Lisa Shalizi, and Robert Haslinger. 'Quantifying self-organization with optimal predictors'. *In Physical Review Letters*, volume 93(11), p. 118701 (2004).

Shapiro, Linda G. and Robert M. Haralick. 'Structural Descriptions and Inexact Matching'. *In Pattern Analysis and Machine Intelligence, IEEE Transactions on*, volume PAMI-3(5), pp. 504 –519 (sept. 1981).

Shehory, Onn and Sarit Kraus. 'Methods for task allocation via agent coalition formation'. *In Artificial Intelligence*, volume 101(1-2), pp. 165–200 (1998).

Sierra, Carles. 'Agent-Mediated Electronic Commerce'. *In Autonomous Agents and Multi-Agent Systems*, volume 9(3), pp. 285–301 (2004).

Smathers, D. and A. Akhil. 'Operating Environment and Functional Requirements for Intelligent Distributed Control in the Electric Power Grid'. Technical report, Sandia National Laboratories (2000).

Smith, N.J. and A.P. Sage. 'An introduction to hierarchical systems theory'. *In Computers & Electrical Engineering*, volume 1(1), pp. 55 – 71 (1973).

Smith, R.G. 'The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver'. *In IEEE Transactions on Computers*, volume 100(12), pp. 1104–1113 (1980).

Steghöfer, Jan-Philipp, Rolf Kiefhaber, Karin Leichtenstern, Yvonne Bernard, Lukas Klejnowski, Wolfgang Reif, Theo Ungerer, Elisabeth André, Jörg Hähner, and Christian Müller-Schloer. 'Trustworthy Organic Computing Systems: Challenges and Perspectives'. *In Autonomic and Trusted Computing– 7th International Conference, ATC 2010, October 26-29, 2010. Proceedings*, volume 6407 of *Lecture Notes in Computer Science*, edited by Bing Xie, Jürgen Branke, S. Sadjadi, Daqing Zhang, and Xingshe Zhou, pp. 62–76. Springer Berlin / Heidelberg, Xian, China (2010).

Steghöfer, Jan-Philipp and Wolfgang Reif. 'Die Guten, die Bösen und die Vertrauenswürdigen – Vertrauen im Organic Computing'. *In Informatik-Spektrum*, volume 35(2), pp. 119–131 (2012).

Steghöfer, Jan-Philipp, Gerrit Anders, Florian Siefert, and Wolfgang Reif. 'A System of Systems Approach to the Evolutionary Transformation of Power Management Systems'. *In Proceedings of INFORMATIK 2013 – Workshop on "Smart Grids"*, Lecture Notes in Informatics. Bonner Köllen Verlag (2013a).

Steghöfer, Jan-Philipp, Pascal Behrmann, Gerrit Anders, Florian Siefert, and Wolfgang Reif. 'HiSPADA: Self-Organising Hierarchies for Large-Scale Multi-Agent Systems'. *In ICAS 2013, The Ninth International Conference on Autonomic and Autonomous Systems*, pp. 71–76. IARIA, Lisbon, Portugal (March 2013b).

Steghöfer, Jan-Philipp, Benedikt Eberhardinger, Florian Nafz, and Wolfgang Reif. 'Synthesis of Observers for Autonomic Evolutionary Systems from Requirements Models'. *In Proceedings of the 6th International Workshop on Distributed Autonomous Network Management Systems*, pp. 1–6. IEEE Computer Society, Washington, D.C., Gent, Belgium (May 2013c).

Sterritt, Roy, Manish Parashar, Huaglory Tianfield, and Rainer Unland. 'A concise introduction to autonomic computing'. *In Adv. Eng. Inform*, volume 19(3), pp. 181–187 (2005).

Núñez Suárez, Jorge, Donie O'Sullivan, Henri Brouchoud, Patrice Cros, Clair Moore, and Ciara Byrne. 'Experiences in the use of FIPA agent technologies for the development of a personal travel application'. *In Proceedings of the fourth international conference on Autonomous agents*, AGENTS '00, pp. 357–364. ACM, New York, NY, USA (2000).

Sudeikat, Jan, Jan-Philipp Steghöfer, Hella Seebach, Wolfgang Reif, Wolfgang Renz, Thomas Preisler, and Peter Salchow. 'Design and Simulation of a Wave-like Self-Organization Strategy for Resource-Flow Systems'. *In Proceedings of The Multi-Agent Logics, Languages, and Organisations Federated Workshops (MALLOW 2010), August 30 – September 2, 2010*, volume 627 of *CEUR Workshop Proceedings*, edited by Olivier Boissier, Amal El Fallah-Seghrouchni, Salima Hassas, and Nicolas Maudet. CEUR-WS.org, Lyon, France (2010).

———. 'On the combination of top-down and bottom-up methodologies for the design of coordination mechanisms in self-organising systems'. *In Information and Software Technology*, volume 54(6), pp. 593–607 (2012).

Tarjan, Robert Endre. 'Edge-disjoint spanning trees and depth-first search'. *In Acta Informatica*, volume 6, pp. 171–185 (1976).

Tran, Quynh-Nhu Numi and Graham Low. 'MOBMAS: A methodology for ontology-based multi-agent systems development'. *In Inf. Softw. Technol.*, volume 50(7-8), pp. 697–722 (June 2008).

Tran, Quynh-Nhu Numi and Graham C Low. *Agent-oriented methodologies*, chapter Comparison of ten agent-oriented methodologies, pp. 341–367. Idea Group, Hershey, PA (2005).

Tran, Quynh-Nhu Numi, Graham Low, and Mary-Anne Williams. 'A preliminary comparative feature analysis of multi-agent systems development methodologies'. *In Proceedings of the 6th international conference on Agent-Oriented Information Systems II*, AOIS'04, pp. 157–168. Springer-Verlag, Berlin, Heidelberg (2005).

Truyen, Frank. 'The Fast Guide to Model Driven Architecture The Basics of Model Driven Architecture'. Technical report, Cephas Consulting Corp. (2006).

Turoff, Murray. 'Past and future emergency response information systems'. *In Communications of the ACM*, volume 45(4), pp. 29–32 (April 2002).

UCTE. 'UCTE Operation Handbook – Policy 1: Load-Frequency Control and Performance'. Technical Report UCTE OH P1, Union for the Co-ordination of Transmission of Electricity (2009).

Valev, Ventzeslav. 'Set partition principles revisited'. *In Advances in Pattern Recognition*, volume 1451 of *Lecture Notes in Computer Science*, pp. 875–881. Springer Berlin / Heidelberg (1998).

Van Zandt, Timothy. 'Hierarchical computation of the resource allocation problem'. *In European Economic Review*, volume 39(3-4), pp. 700–708 (April 1995).

Vasalou, Asimina and Jeremy Pitt. 'Reinventing Forgiveness: A Formal Investigation of Moral Facilitation'. *In Trust Management*, volume 3477 of *Lecture Notes in Computer Science*, edited by Peter Herrmann, Valérie Issarny, and Simon Shiu, pp. 39–90. Springer Berlin / Heidelberg (2005).

Vytelingum, P., Sarvapali D. Ramchurn, T.D. Voice, Alex Rogers, and Nick R. Jennings. 'Trading agents for the smart electricity grid'. *In Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, volume 1, pp. 897–904. International Foundation for Autonomous Agents and Multiagent Systems (2010).

Weaver, W. 'Science and complexity'. *In American Scientist*, volume 36(4) (1948).

White, Steve R., James E. Hanson, Ian Whalley, David M. Chess, Alla Segal, and Jeffrey O. Kephart. 'Autonomic computing: Architectural approach and prototype'. *In Integr. Comput.-Aided Eng*, volume 13(2), pp. 173–188 (2006).

Wilensky, Uri. 'NetLogo' (1999).
URL: *http://ccl.northwestern.edu/netlogo/*

Wooldridge, Michael. *An Introduction to Multiagent Systems*. Wiley. com (2008).

Xiao, Xue and Zhang XueYan. 'The method engineering process for multi-agent system development'. *In Proceedings of the 1st International Conference on Forensic Applications and Techniques in Telecommunications, Information, and Multimedia*, e-Forensics '08, pp. 42:1–42:4. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium (2008).

Yokoo, Makoto. *Distributed Constraint Satisfaction.* Springer Series on Agent Technology. Springer Berlin Heidelberg (2001).

Younis, Ossama and Sonia Fahmy. 'HEED: A Hybrid, Energy-Efficient, Distributed Clustering Approach for Ad Hoc Sensor Networks'. *In IEEE Transactions on Mobile Computing*, volume 3, pp. 366–379 (2004).

Zafra-Cabeza, A., M.A. Ridao, I. Alvarado, and E.F. Camacho. 'Applying risk management to combined heat and power plants'. *In Power Systems, IEEE Transactions on*, volume 23(3), pp. 938–945 (2008).

Zambonelli, Franco, NicholasR. Jennings, and Michael Wooldridge. 'Organisational Abstractions for the Analysis and Design of Multi-agent Systems'. *In Agent-Oriented Software Engineering*, volume 1957 of *Lecture Notes in Computer Science*, edited by Paolo Ciancarini and MichaelJ. Wooldridge, pp. 235–251. Springer Berlin Heidelberg (2001).

Zhang, B., P.B. Luh, E. Litvinov, T. Zheng, F. Zhao, J. Zhao, and C. Wang. 'Electricity auctions with intermittent wind generation'. *In Power and Energy Society General Meeting*, pp. 1–8. IEEE (2011).

Zhang, Shu and Malcolm R. Irving. 'Analytical algorithm for constraint relaxation in LP-based optimal power flow'. *In IEE Proceedings – Generation, Transmission and Distribution*, volume 140(4), pp. 326–330 (1993).

Own Peer-Reviewed Publications

Anders, Gerrit, Hella Seebach, Florian Nafz, Jan-Philipp Steghöfer, and Wolfgang Reif. 'Decentralized Reconfiguration for Self-Organizing Resource-Flow Systems Based on Local Knowledge'. *In 8th IEEE Int. Conference on Engineering of Autonomic and Autonomous Systems (EASe)*, pp. 20–31 (2011).

Anders, Gerrit, Jan-Philipp Steghöfer, Florian Siefert, and Wolfgang Reif. 'Patterns to Measure and Utilize Trust in Multi-agent Systems'. *In Self-Adaptive and Self-Organizing Systems Workshops (SASOW), 2011 Fifth IEEE Conference on*, pp. 35 –40. IEEE Computer Society, Washington, D.C. (Oct. 2011).

Anders, Gerrit, Florian Siefert, Jan-Philipp Steghöfer, and Wolfgang Reif. 'A Decentralized Multi-agent Algorithm for the Set Partitioning Problem'. *In PRIMA 2012: Principles and Practice of Multi-Agent Systems*, volume 7455 of *Lecture Notes in Computer Science*, edited by Iyad Rahwan, Wayne Wobcke, Sandip Sen, and Toshiharu Sugawara, pp. 107–121. Springer Berlin / Heidelberg (2012).

———. 'Trust-Based Scenarios – Predicting Future Agent Behavior in Open Self-Organizing Systems'. *In Proc. of the 7th Int. Workshop on Self-Organizing Systems (IWSOS 2013)*. Springer (May 2013).

Anders, Gerrit, Jan-Philipp Steghöfer, Florian Siefert, and Wolfgang Reif. 'A Trust- and Cooperation-Based Solution of a Dynamic Resource Allocation Problem'. *In Seventh IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*. IEEE Computer Society, Washington, D.C., Philadelphia, PA (September 2013).

Eberhardinger, Benedikt, Jan-Philipp Steghöfer, Florian Nafz, and Wolfgang Reif. 'Model-driven Synthesis of Monitoring Infrastructure for Reliable Adaptive Multi-Agent Systems'. *In Proceedings of the 24th IEEE International Symposium on Software Reliability Engineering (ISSRE 2013)*. IEEE Computer Society, Washington, D.C., Pasadena, CA (November 2013).

von Mammen, Sebastian and Jan-Philipp Steghöfer. 'Self-organized middle-out abstraction of complexity'. *In Awareness Magazine* (June 2013). Available online.

———. 'Bring it on, Complexity! Present and future of self-organising middle-out abstraction'. *In The Computer After Me*, edited by Jeremy Pitt, pp. 69–88. Imperial College Press (2014).

von Mammen, Sebastian, Jan-Philipp Steghöfer, Jörg Denzinger, and Christian Jacob. 'Self-organized Middle-Out Abstraction'. *In Self-Organizing Systems*, volume 6557 of *Lecture Notes in Computer Science*, edited by Christian Bettstetter and Carlos Gershenson, pp. 26–31. Springer Berlin Heidelberg (2011).

Nafz, Florian, Frank Ortmeier, Hella Seebach, Jan-Philipp Steghöfer, and Wolfgang Reif. 'Implementing Organic Computing Systems with AgentService'. *In ENASE 2008 – Proceedings of the 3rd International Conference on Evaluation of Novel Approaches to Software Engineering, May 4-7, 2008*, edited by Cesar Gonzalez-Perez and Stefan Jablonski, pp. 64–71. INSTICC Press, Funchal, Madeira, Portugal (2008).

———. 'A generic software framework for role-based Organic Computing systems'. *In 2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2009, May 18-19, 2009*, pp. 96–105. IEEE, Vancouver, BC, Canada (2009a).

———. 'A Universal Self-Organization Mechanism for Role-Based Organic Computing Systems'. *In Autonomic and Trusted Computing, 6th International Conference, ATC 2009, July 7-9, 2009, Proceedings*, volume 5586 of *Lecture Notes in Computer Science*, edited by Wolfgang Reif, Guojun Wang, and Jadwiga Indulska, pp. 17–31. Springer, Brisbane, Australia (2009b).

Nafz, Florian, Hella Seebach, Jan-Philipp Steghöfer, Simon Bäumler, and Wolfgang Reif. 'A Formal Framework for Compositional Verification of Organic Computing Systems'. *In Autonomic and Trusted Computing – 7th International Conference, ATC 2010, October 26-29, 2010. Proceedings*, volume 6407 of *Lecture Notes in Computer Science*, edited by Bing Xie, Jürgen Branke, Seyed Masoud Sadjadi, Daqing Zhang, and Xingshe Zhou, pp. 17–31. Springer Berlin / Heidelberg, Xi'an, China (2010).

Nafz, Florian, Hella Seebach, Jan-Philipp Steghöfer, Gerrit Anders, and Wolfgang Reif. 'Constraining Self-organisation Through Corridors of Correct Behaviour: The Restore Invariant Approach'. *In Organic Computing – A Paradigm Shift for Complex Systems*, volume 1 of *Autonomic Systems*, edited by Christian Müller-Schloer, Hartmut Schmeck, and Theo Ungerer, pp. 79–93. Springer Basel (2011).

Nafz, Florian, Jan-Philipp Steghöfer, Hella Seebach, and Wolfgang Reif. 'Formal Modeling and Verification of Self-Organizing Systems Based on Observer/Controller-Architectures'. *In Assurances for Self-Adaptive Systems*, volume 7740 of *Lecture Notes in Computer Science*, edited by J. Cámara, R. de Lemos, C. Ghezzi, and A. Lopes, pp. 80–111. Springer Berlin / Heidelberg (2013).

Schiendorfer, Alexander, Jan-Philipp Steghöfer, Alexander Knapp, and Wolfgang Reif. 'Constraint Relationships for Soft Constraints'. *In Proceedings of the 33rd SGAI International Conference on Artificial Intelligence*. Springer (December 2013).

Seebach, Hella, Florian Nafz, Jan-Philipp Steghöfer, and Wolfgang Reif. 'A Software Engineering Guideline for Self-Organizing Resource-Flow Systems'. *In IEEE International Conference on Self-Adaptive and Self-Organizing System (SASO)*, pp. 194–203. IEEE Computer Society, Washington, D.C., Budapest, Hungary (2010).

———. 'How to Design and Implement Self-organising Resource-Flow Systems'. *In Organic Computing – A Paradigm Shift for Complex Systems*, edited by Christian Müller-Schloer, Hartmut Schmeck, and Theo Ungerer, Autonomic Systems, pp. 145–161. Springer Basel (2011).

Steghöfer, Jan-Philipp and Wolfgang Reif. 'Die Guten, die Bösen und die Vertrauenswürdigen – Vertrauen im Organic Computing'. *In Informatik-Spektrum*, volume 35(2), pp. 119–131 (2012).

Steghöfer, Jan-Philipp, Jörg Denzinger, Holger Kasinger, and Bernhard Bauer. 'Improving the Efficiency of Self-Organizing Emergent Systems by an Advisor'. *In Engineering of Autonomic and Autonomous Systems (EASe), 2010 Seventh IEEE International Conference and Workshops on*, pp. 63–72. IEEE Computer Society, Washington, D.C., Oxford, UK (2010).

Steghöfer, Jan-Philipp, Rolf Kiefhaber, Karin Leichtenstern, Yvonne Bernard, Lukas Klejnowski, Wolfgang Reif, Theo Ungerer, Elisabeth André, Jörg Hähner, and Christian Müller-Schloer. 'Trustworthy Organic Computing Systems: Challenges and Perspectives'. *In Autonomic and Trusted Computing– 7th International Conference, ATC 2010, October 26-29, 2010. Proceedings*, volume 6407 of *Lecture Notes in Computer Science*, edited by Bing Xie, Jürgen Branke, S. Sadjadi, Daqing Zhang, and Xingshe Zhou, pp. 62–76. Springer Berlin / Heidelberg, Xian, China (2010).

Steghöfer, Jan-Philipp, Pratik Mandrekar, Florian Nafz, Hella Seebach, and Wolfgang Reif. 'On Deadlocks and Fairness in Self-organizing Resource-Flow Systems'. *In Architecture of Computing Systems - ARCS 2010, 23rd International Conference, February 22-25, 2010. Proceedings*, volume 5974 of *Lecture Notes in Computer Science*, edited by Christian Müller-Schloer, Wolfgang Karl, and Sami Yehia, pp. 87–100. Springer, Hannover, Germany (2010a).

Steghöfer, Jan-Philipp, Florian Nafz, Wolfgang Reif, Yvonne Bernard, Lukas Klejnowski, Jörg Hähner, and Christian Müller-Schloer. 'Formal Specification and Analysis of Trusted Communities'. *In Self-Adaptive and Self-Organizing Systems Workshop (SASOW), 2010 Fourth IEEE International Conference on*, pp. 190–195. IEEE Computer Society, Washington, D.C., Budapest, Hungary (2010b).

Steghöfer, Jan-Philipp, Gerrit Anders, Florian Siefert, and Wolfgang Reif. 'A System of Systems Approach to the Evolutionary Transformation of Power Management Systems'. *In Proceedings of INFORMATIK 2013 – Workshop on "Smart Grids"*, Lecture Notes in Informatics. Bonner Köllen Verlag (2013).

Steghöfer, Jan-Philipp, Pascal Behrmann, Gerrit Anders, Florian Siefert, and Wolfgang Reif. 'HiSPADA: Self-Organising Hierarchies for Large-Scale Multi-Agent Systems'. *In ICAS 2013, The Ninth International Conference on Autonomic and Autonomous Systems*, pp. 71–76. IARIA, Lisbon, Portugal (March 2013).

Steghöfer, Jan-Philipp, Benedikt Eberhardinger, Florian Nafz, and Wolfgang Reif. 'Synthesis of Observers for Autonomic Evolutionary Systems from Requirements Models'. *In Proceedings of the 6th International Workshop on Distributed Autonomous Network Management Systems*, pp. 1–6. IEEE Computer Society, Washington, D.C., Gent, Belgium (May 2013).

Sudeikat, Jan, Jan-Philipp Steghöfer, Hella Seebach, Wolfgang Reif, Wolfgang Renz, Thomas Preisler, and Peter Salchow. 'Design and Simulation of a Wave-like Self-Organization Strategy for Resource-Flow Systems'. *In Proceedings of The Multi-Agent Logics, Languages, and Organisations Federated Workshops (MALLOW 2010), August 30 – September 2, 2010*, volume 627 of *CEUR Workshop Proceedings*, edited by Olivier Boissier, Amal El Fallah-Seghrouchni, Salima Hassas, and Nicolas Maudet. CEUR-WS.org, Lyon, France (2010).

———. 'On the combination of top-down and bottom-up methodologies for the design of coordination mechanisms in self-organising systems'. *In Information and Software Technology*, volume 54(6), pp. 593–607 (2012).