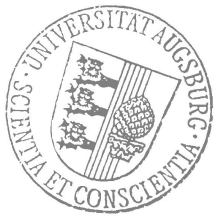
Design and Operation of Efficient Self-Organizing Systems

Dissertation

for the degree of

Doctor of Natural Sciences (Dr. rer. nat.)



Holger Kasinger

University of Augsburg

Department of Computer Science
Programming Distributed Systems Lab

July 2010

Design and Operation of Efficient Self-Organizing Systems

Supervisor:	**Prof. Dr. Bernhard Bauer**, Department of Computer Science University of Augsburg, Germany
Advisors:	**Prof. Dr. Jörg Denzinger**, Department of Computer Science, University of Calgary, Canada
	Prof. Dr. Theo Ungerer, Department of Computer Science, University of Augsburg, Germany
Day of defense:	10. 11. 2010

Augsburg, July 2010

To Melanie, Lukas, and Paul

Abstract

At the beginning of this 21st century, companies are faced with two key challenges with regard to their IT systems: first, the increasing complexity of IT systems gives rise to escalating operational expenditures for their administration and maintenance. Second, the intensifying, worldwide competition in all markets requires IT systems providing more agility, flexibility, scalability, robustness, and adaptivity in tackling daily businesses. Consequently, companies call for IT solutions with a high degree of autonomy – in order to manage themselves – as well as a high degree of decentralization – in order to provide the required, beneficial properties.

Self-organizing emergent systems are generally acknowledged as a potential solution able to cover both of these requirements. They consist of many, simple elements (e. g. agents, servers, mobile devices, or robots), which have only partial or even no global system knowledge and make their decisions solely based on locally available information. The global coherent system behavior is achieved only by means of the local actions and interactions between the elements, each unaware of the system's goals. The problem-solving power of a self-organizing emergent system hence mainly resides in the interactions between its elements instead of the internal reasoning of individual elements.

However, there exist several problems and challenges, which hinder the acceptance of self-organizing emergent systems by industry. This thesis tackles two of them: first, the design of efficient self-organizing emergent systems today is too complex, time-consuming, and costly. Second, an acceptable efficiency of self-organizing emergent systems during the operation cannot be guaranteed for all situations. These problems form two challenging paradoxes: first, in order to conquer system complexity, one has to create more complex systems in a much more complex way. Second, in order to lower operational expenditures, one has to use potentially inefficient systems that actually may increase operational expenditures.

Thus, this thesis presents several artifacts that on the one hand simplify the design of effective as well as efficient self-organizing emergent systems and on the other hand facilitate their efficient operation even in unforeseeable situations. In more detail, the major contributions of this thesis are as follows:

1. We investigate the general principles behind decentralized coordination by means of chemical stimuli (infochemicals) between organisms in biology and adopt them in a computational coordination model. Because infochemical-based coordination (IBC) is the most universally employed communication and coordination model between organisms in biology with a plethora of inspiring examples, the formally adopted principles provide the foundation for the future

specification of various, bio-inspired, decentralized coordination mechanisms using digital infochemicals. The expressiveness of the adopted model and the use of infochemicals with different semantics, dynamics, and functions, allow for the simplified design of more efficient solutions and solution processes compared to existing approaches.

2. We present a corresponding design pattern that encapsulates the adopted coordination model in a systematic way familiar to software engineers. The pattern hides the inherent complexity of the designed system and makes meaningful abstractions from the biological principles. Furthermore, we present design guidelines that support the identification and adaptation of new coordination mechanisms based on IBC. This simplifies the design of new solutions but does not force engineers to be biological experts at the same time. Moreover, we develop an adequate tool for the simulation of various coordination models and mechanisms in different application domains.

3. We present the general model of an advisor that is able to improve the efficiency of self-organizing emergent systems solving dynamic optimization problems with recurring tasks. This so-called Efficiency Improvement Advisor (EIA) realizes an unobtrusive feedback and learning mechanism that is independent of the coordination model or mechanism used by the underlying self-organizing emergent system as well as the problem domain in hand. The EIA in particular takes into account the low observability and poor controllability of self-organizing emergent systems during operation, considers their openness and basic autonomy, and preserves their beneficial self-organizing emergent properties.

4. We develop a decentralized coordination mechanism based on IBC, which takes its inspiration from the pollination of flowers by honey bees. This so-called pollination-inspired coordination (PIC) mechanism demonstrates the many beneficial capabilities of IBC and can e. g. be used for the self-organizing emergent solution to every-day problems in logistics, more specifically pickup and delivery problems. Likewise we develop an instantiation of the EIA model for this mechanism and domain. An experimental evaluation proves the efficiency of the PIC mechanism as well as the achieved improvements by the EIA.

Zusammenfassung

Im Hinblick auf ihre IT-Systeme sind Unternehmen zu Beginn des 21. Jahrhunderts mit zwei entscheidenden Herausforderungen konfrontiert: Zum Einen führt die wachsende Komplexität von IT-Systemen zu permanent steigenden Betriebsausgaben für deren Administration und Wartung. Zum Anderen verlangt der sich weltweit intensivierende Wettbewerb in vielen Märkten nach IT-Systemen, welche mehr Agilität, Flexibilität, Skalierbarkeit, Robustheit und Adaptivität zur Bewältigung der täglichen Geschäfte bieten. Als Konsequenz fordern diese Unternehmen IT-Lösungen mit einem hohen Grad an Autonomie – so dass sich die Systeme selbständig administrieren und warten können – sowie einem hohen Grad an Dezentralität – um die genannten Eigenschaften erbringen können.

Selbst-organisierende emergente Systeme werden allgemein als solch eine Lösung gesehen, welche beide Anforderungen abdeckt. Diese Systeme bestehen aus einer Vielzahl einfacher Elemente, wie z. B. Agenten, Servern, mobiler Geräte, oder Robotern, welche allerdings nur partielles oder gar kein globales Wissen über das System besitzen und ihre Entscheidungen lediglich auf lokal verfügbaren Informationen basierend treffen. Das global kohärente Systemverhalten wird einzig und allein mittels den lokalen Aktionen und Interaktionen zwischen den Systemelementen erreicht, wobei jedes einzelne Element in Unkenntnis über die Aufgaben und Ziele des gesamten Systems ist. Die problemlösende Fähigkeiten dieser Systeme liegen daher hauptsächlich in den Interaktionen zwischen allen Elementen anstatt im internen Denken und Folgern individueller Elemente.

Allerdings existieren mehrere Probleme und Herausforderungen, welche heutzutage die breite Akzeptanz solcher Systeme seitens der Industrie verhindern. Diese Arbeit befasst sich mit zwei dieser Probleme: Zum Einen ist das Design von effizienten, selbst-organisierenden emergenten Systemen heutzutage zu komplex, zeitintensiv und kostspielig. Zum Anderen kann eine akzeptable Effizienz dieser Systeme während ihres Betriebs nicht immer garantiert werden. Diese Probleme stellen zwei herausfordernde Paradoxe dar: Erstens, um die Auswirkungen der steigenden Komplexität von IT-Systemen zu reduzieren, müssen noch komplexere Systeme auf eine viel komplexere Art und Weise entwickelt werden. Zweitens, um Betriebsausgaben senken zu können, müssen potentiell ineffiziente Systeme eingesetzt werden, welche möglicherweise die Betriebsausgaben gar steigern.

Daher präsentiert diese Arbeit mehrere Artefakte, welche einerseits das Design von effektiven als auch effizienten, selbst-organisierenden emergenten Systemen vereinfachen und andererseits deren effizienten Betrieb selbst in vorher nicht bedachten Situation ermöglichen. Genauer gesagt enthält diese Arbeit folgende Hauptbeiträge zur Bewältigung der genannten Probleme:

1. Wir untersuchen die allgemeinen Prinzipien der dezentralen Koordination mittels chemischen Duftstoffen (sog. Infochemikalien) zwischen biologischen Organismen und adoptieren diese in einem Koordinationsmodell für IT-Systeme. Da die Koordination mittels Infochemikalien das am weitesten verbreitete Kommunikations- und Koordinationsmodell zwischen Organismen ist und daher eine Fülle an inspirierenden Beispielen bereithält, bieten die adoptierten Prinzipien die Grundlage für die zukünftige Spezifikation verschiedener, biologisch-inspirierter, dezentraler Koordinationsmechanismen basierend auf digitalen Infochemikalien. Die Ausdrucksfähigkeit des entwickelten Koordinationsmodells und die Verwendung von Infochemikalien mit unterschiedlicher Semantik, Dynamik und Funktionalität ermöglichen ein einfacheres Design von effizienteren Lösungen und Prozessen im Vergleich zu existierenden Ansätzen.

2. Wir entwickeln ein Design Pattern, welches das abgeleitete Koordinationsmodell in einer für Softwareingenieure bekannten und systematischen Art und Weise kapselt. Das Pattern blendet die inhärente Komplexität des entwickelten Systems aus und vollzieht sinnvolle Abstraktionen von den biologischen Prinzipien. Dazu passend entwickeln wir Richtlinien für das Design, welche die Identifikation und Adaptation von neuen, auf digitalen Infochemikalien basierenden Koordinationsmechanismen unterstützen. Beides vereinfacht das Design neuer Lösungen, aber verlangt von Softwareingenieuren nicht gleichzeitig auch noch Experten auf dem Gebiet der Biologie zu sein. Zur weiteren Unterstützung entwickeln wir ein Tool zur Simulation verschiedener Koordinationsmodelle und -mechanismen für unterschiedliche Anwendungsgebiete.

3. Wir entwickeln das generelle Modell eines sog. Advisors (Ratgeber), welcher in der Lage ist, die Effizienz von selbst-organisierenden emergenten Systemen bei der Lösung von dynamischen Optimierungsproblemen mit wiederkehrenden Aufgaben zur Laufzeit zu verbessern. Das Modell realisiert einen diskreten Feedback- und Lernmechanismus, welcher unabhängig von dem verwendeten Koordinationsmodell oder -mechanismus sowie der Problemdomäne ist. Das Modell des Advisors berücksichtigt insbesondere die geringe Beobachtbarkeit und schlechte Kontrollierbarkeit von selbst-organisierenden emergenten Systemen während ihres Betriebs als auch ihre Offenheit und die grundlegende Autonomie ihrer Elemente. Zudem bewahrt das Modell die vorteilhaften, selbst-organiserenden, emergenten Eigenschaften dieser Systeme.

4. Wir entwickeln einen dezentralen Koordinationsmechanismus basierend auf digitalen Infochemikalien, welcher seine Inspiration aus der Betäubung von Blüten durch Honigbienen gewinnt. Dieser Mechanismus kann bspw. für die selbst-organisierende emergente Lösung von alltäglichen Problemen in der Logistik eingesetzt werden. Gleichermaßen instantiieren wir das Modell des Advisors für diesen Mechanismus und diese Domäne. Eine experimentelle Evaluierung beweist die Effizienz des entwickelten Koordinationsmechanismus sowie die erzielten Verbesserungen der Effizienz durch den Advisor.

Acknowledgments

At this point I thank all the people that supported me in writing this thesis. First of all I thank my supervisor Prof. Dr. Bernhard Bauer for giving me the opportunity to conduct my research in the area of self-adaptive and self-organizing systems and for giving me the freedom to develop the topic of this thesis according to my own ideas. Not only his friendly guidance over the last years as well as his financial support for lots of travels were the basis for the successful completion of this thesis.

I am also deeply indebted to my advisor Prof. Dr. Jörg Denzinger for his valuable comments and the many scientific discussions we had during my stays in Calgary. In particular his personal dedication in showing my the scenic and culinary advantages of Alberta made me gladly come back to Calgary time and again.

I thank Prof. Dr. Theo Ungerer, who also accepted to be an advisor of my thesis.

I am very grateful to my current and prior colleagues at the Programming Distributed Systems Lab for a friendly and cheerful working atmosphere as well as various after-work activities. Sharing individual problems as well as discussing novel solutions have made my day-to-day work balanced and varied. I also thank all diploma students and student workers that assisted me in trying several directions and approaches as well as helped me to save time with their implementations.

Special thanks go to my parents Frigga and Knut for their education and support during my schooldays and studies. They have always encouraged me and my brother to do the best in all matters of life.

Last but not least I thank especially my wife Melanie for her understanding for my work and her enduring patience for months of traveling. Without her unconditional love and her bringing up of our two children Lukas and Paul this thesis would never have been finished.

Contents

Part I

Self-Organizing Emergent Systems

Chapter 1

Introduction

The beginning of the 21st century is dominated by computer systems and environments that comprise complex, heterogeneous tangles of hardware, middleware, and software forming nebulous communication structures as well as arcane system architectures. The integration and configuration of new components into these systems is a time-consuming and error-prone task, the operation and maintenance of these systems requires vast quantities of human and monetary resources, and the combination and reuse of different (sub)systems is extremely challenging. For instance, most organizations typically spend about three-fourths of their application deployment time and costs on the integration of different systems [Mur04] and one third to one half of their IT budget on preventing or recovering from crashes [GKM02].

At the present rate of growth of computer systems' size and complexity, in near future even skilled IT professionals may fail to manage these computer systems and environments properly [Kep05]. Due to these facts, companies are more and more forced to spend a progressively growing rate of their operational expenditures (OPEX) on managing their computer systems instead on managing their core businesses. A few years ago, the total cost of ownership (TCO) of computer system installations already outstripped purchase costs by a factor of 3 to 18, depending on the type of system [GKM02]. Thus, today CIOs and CTOs are tasked with reducing TCO, in particular OPEX for the management of their computer systems [Mur04].

However, more and more companies become aware of the fact that conventional paradigms for the management of their computer systems are particularly challenged in tackling this complexity and that new and efficient ways to manage these systems have to be found. Consequently, a number of industrial initiatives have been launched by major IT vendors, e. g. IBM's Autonomic Computing [IBM02], Intel's Proactive Computing [Ten00], Microsoft's Dynamic Systems Initiative [Mic07], Hewlett Packard's Adaptive Infrastructure [Hew08], or Sun's N1 [Sun02b], all addressing these challenges from the business perspective. But also on the academic side comparable initiatives have been launched, e. g. Autonomic Communication [Aut04, DDF+06] or Organic Computing [OCI05]. Across the board all initiatives agree that a kind of self-management of computer systems and networks, i. e. a high degree of autonomy and autonomicity, offers the most promising approach in order to cope with the increasing complexity. Future computer systems have to be able to adapt at runtime to changing user needs, system intrusions or faults, changing operational environments, and resource variability, while keeping most of their com-

plexity hidden from the user or administrator. They have to manage themselves, i. e. their tasks, processes, processors, software, storage, networks, devices, and all other components, as far as appropriate, in accordance with high-level objectives specified by humans [KC03]. As a result, self-managing respectively self-adaptive systems are more versatile, resilient, dependable, recoverable, customizable, and configurable, which will consequently reduce TCO. Such properties are very often also referred to as self-* properties [Hor01], e. g. self-configuring, self-optimizing, self-healing, and self-protecting.

Beside the call for self-managing computer systems, the intensifying, worldwide competition in almost all markets additionally calls for computer systems that provide more agility, flexibility, scalability, robustness, and adaptivity in tackling everyday business. As a result, future computer systems have to exhibit a higher degree of distribution and decentralization compared to today's systems, lacking any global or central control. The vast majority of future computer systems thus will be characterized by context-awareness, openness, locality in control, and locality in interactions [ZP03]. This trend is already observable by the converging activities from several research areas, such as distributed artificial intelligence (DAI) respectively multi-agent systems [Jen01], Ubiquitous Computing [Wei91], or peer-to-peer (P2P) Computing [RIF02]. Consequently, the elements of future computer systems are compelled to efficiently self-organize their actions and interactions [JBL06] as well as to build emergent properties [DH04] in order to fulfill tasks along with the required high degree of autonomy and the favored self-* properties.

Although prototypical self-organizing emergent (computer) systems already have been applied to different case studies in e. g. traffic control, network topology management, timetable scheduling, intrusion detection systems, manufacturing control, packet delivery services, or mobile ad-hoc networks (cf. [MRF+03, BSKN05, MZ06, BSHZ06, BHJY07]), there still exist a couple of challenging problems that are opposed to their widespread application. This thesis tackles two of these problems, which are affiliated to the design as well as the operation phase of the systems' life cycle: first, the design of self-organizing emergent systems today is too complex, time-consuming, and costly in order to engineer effective but yet efficient solutions for various problem domains; second, a required efficiency of these solutions during operation can not be guaranteed, in particular not for dynamic problems. Section 1.1 explains these problems along with their challenges in more detail.

The objectives and contributions of this thesis hence are twofold. On the one hand, this thesis simplifies the design of self-organizing emergent systems by developing several artifacts that can be used for the systematic engineering of effective and at the same time efficient solutions. On the other hand, this thesis facilitates an efficient operation of self-organizing emergent systems by developing an approach that autonomously adapts the local behavior of self-organizing elements in order to improve the efficiency of the global solution in certain situations. Section 1.2 explains the objectives, the approaches taken, and the contributions in more detail. Finally, Section 1.3 presents an outline of the chapters of this thesis, while Section 1.4 lists publications in which parts of this thesis have been previously published.

1.1 Problems and Challenges

'Conquer system complexity' is one of the five grand research challenges in information systems at the beginning of the 21st century, according to the US Computer Research Association [Com02]: *"Meeting this challenge requires a reformulation at all levels of computer architecture, software organization, and system design to break through the complexity barrier and create more robust systems. We must be able to design and implement systems that can autonomously adapt, maintain, repair, and heal themselves. Such innovations will substantially lower the total cost of ownership, reduce the need for intense manual supervision, and increase the future reliability and scalability of our systems."* The grand research challenges 'Build systems you can count on' [Com02] as well as 'Dependable systems evolution' – formulated by the UK Computing Research Committee [KH08] – similarly address systems in demand of a high degree of autonomy.

The usage of the concepts self-organization and emergence for the realization of such autonomous systems and their self-* properties is a promising but demanding approach (cf. e. g. [ERA+03, DH04, JBL06, ABI07, Ant08]). A self-organizing emergent system[1] consists of many locally interacting elements, which can either be autonomous software elements, such as agents, or autonomous real-world elements with computing and networking capabilities, such as servers, mobile devices, robots, or cars. The elements have only partial or even no global system knowledge and determine their actions solely based on local information available from their neighbors in the communication topology as well as the environment. Because the elements normally are kept relatively simple, e. g. for scalability reasons or due to limited resources, single elements cannot direct such a complex system toward a global goal or behavior on their own. Instead, the global coherent system behavior (on the macroscopic level) is achieved only by means of the local actions and interactions between the elements (on the microscopic level), each unaware of the systems' goals. The problem-solving power of a self-organizing emergent system hence mainly resides in the interactions between its elements instead of the internal reasoning of individual elements.

Self-organizing emergent computer systems mostly function in the same way as observable examples do in contexts such as biology, ecology, chemistry, physics, economics, or sociology. Hence, instead of achieving efficiency, optimality, and predictability, new properties such as scalability, robustness to failures, flexibility w. r. t. system changes, and adaptivity w. r. t. environmental changes are achieved, yielding advantageous self-* properties. However, among other things there are two problems that prevent self-organizing emergent systems from being practically applicable for the effective and efficient usage in various areas.

[1] This class of computer systems is sometimes also referred to as decentralized autonomic computing (DAC) system, complex adaptive system (CAS), or just self-organizing system (SOS).

Problem 1: Design of Efficient Systems is Too Complex, Time-consuming, and Costly

The design of conventional computer systems depends fundamentally on the assumption that any (sub)system can be described wholly by describing the behavior of its parts and their interactions (cf. [Som01]). In contrast, the pathway from element behavior to system behavior and vice versa is not clear for self-organizing emergent systems. A profound model that relates the microscopic level, i. e. the local behaviors of the individual elements, to the macroscopic level of organizations, societies, or systems does not exist yet (cf. e. g. [AGMS87, KR02, Saw03, And04]). This vague coherence, which is often referred to as the 'micro-macro gap', makes the design of efficient self-organizing emergent systems complex, time-consuming, and costly. The solution of this problem bears several challenges:

- **Challenge 1 – Provide versatile and coherent models for efficient decentralized coordination:** An essential key for the reasonable design of efficient self-organizing emergent solutions is the adequate coordination of the system elements' local actions and interactions (cf. [DH05, SGK06]). However, conventional coordination models are inadequate for dealing with the challenges coming along with these solutions, such as facilitating adaptivity, robustness, scalability, etc. Even more, because today only a few models for decentralized coordination exist, very often the required solution of a problem in hand exceeds the capability of such a model, so that the functionality and efficiency, which are not provided by the model, have to be integrated respectively improved in a time-consuming ad hoc manner. The problem in hand may even force an engineer to use several decentralized coordination models in parallel, because every model has to play a part in contributing to the solution (see e. g. [DH07b, WHHS09]). This in turn requires high efforts for horizontally integrating the required coordination models and infrastructures, respectively.

- **Challenge 2 – Specify efficient coordination mechanisms that fulfill global system requirements:** It is not obvious how to design and build a system we do not even fully understand, in more detail, how to specify the local behaviors of system elements such that the system as a whole demonstrates a coherent global behavior that fulfills the global system requirements, such as minimum throughputs or maximal waiting times. However, a reasonable design of self-organizing emergent solutions requires either to re-use existing coordination mechanisms, see e. g. [SR08], or to specify new coordination mechanisms employing simple behavioral and interaction rules for the system elements that efficiently achieve *"functions that are useful to the system's stakeholders"* [PB04], *"the required macroscopic behavior"* [De 07], *"the right behavior at the global level"* [GCGC08]. The specification should not require an engineer to be a biological, economical, physical, and social expert all at once. To the contrary, it should be based on and guided by the expressiveness of the used coordination model.

- **Challenge 3 – Provide design patterns, guidelines, and tools:** A time-saving and cost-effective design of efficient self-organizing emergent systems requires design patterns that capture recurring solutions to standard problems. Such patterns have to hide the inherent complexity of the designed system and focus on environment characterizations, local behaviors, and realized self-* properties. In addition, design guidelines are required that support engineers in specifying new coordination mechanisms based on the used coordination models. Similarly, simulation tools are required that support engineers in identifying and selecting the most suitable and efficient coordination model(s) and mechanism(s) according to the system requirements.

Please note that there are two different dimensions of efficiency that have to be considered. On the one hand, the *solution process* formed by a self-organizing emergent system has to be efficient in terms of messages sent, coordination and communication overhead, etc. On the other hand, the *solution* produced by a self-organizing emergent system has to be efficient in terms of time, costs, required resources, etc. depending on the problem and application domain. Both dimensions have to be carefully addressed, which leads us to the second problem tackled by this thesis.

Problem 2: Efficiency During Operation Can Not be Guaranteed

Apart from the advantageous properties such as scalability, robustness, flexibility, and adaptivity inherently realized by self-organizing emergent systems, certain application domains, in particular in industrial settings, nonetheless call for the achievement and maintenance of a certain degree of efficiency by these systems, even while they solve highly dynamic, complex, and often unpredictable problems. Approaches able to (partly) guarantee a required efficiency already at design time, which are mostly based on extensive simulations prior to the deployment (e. g. [BGP06, GVCO08, SHW08]) but also on elaborated design methodologies [DH05], interactive verification [HRS91], model checking [CGP99], or formal modeling [RS06], are mostly insufficient, because self-organizing emergent systems are expected to function in open and very dynamic environments with unforeseeable contingencies, i. e. changes may be too complex or too frequent to be completely constrained or predicted in advance. A major issue thereby is the fact that the problems, in more detail the tasks that have to be fulfilled by the system elements in a self-organizing manner, usually change dynamically.

Therefore, solving such dynamic problems by self-organizing emergent systems optimally requires on the one hand as optimal local decisions as possible and on the other hand appropriate adaptations of the system's global structure or behavior, preferably maintaining a high degree of autonomy. In order to make 'optimal' local decisions on its own, a system element would have to be in possession of an abundance of relevant information. This includes information about the environment topology (e. g. networks, machines, customers, etc.), the current and future state

of the environment, in particular the problem-relevant tasks, and the current and future intended behavior of other elements. This would not only force the elements to quickly gather real-time information from a large number of (possibly unknown) entities, but also to be able to 'look into the future', such that a dynamically appearing task can be assigned to the best system element (with respect to global optimality of the solution), while other tasks are already executed by the elements. Apparently, such an approach to gain almost global knowledge constitutes a very complex endeavor.

Thus, approaches and system architectures are required that are able to improve the efficiency of self-organizing emergent systems during operation, even in unforeseen and unexpected situations. In other words, these systems have to assess their own behavior during operation and to change their behavior or structure when the assessment indicates that a better performance is possible. This is essentially the essence of self-adaptive systems (cf. [CGI+09]). In case of conventional computer systems usually an additional subsystem is added as a controller for the purpose of self-adaptation, which realizes a closed feedback control loop. The controller monitors and analyzes the underlying basic system and adapts the structure or behavior of this system based on high-level objectives specified by humans. In case of self-organizing emergent systems, the integration of such a controller in general bears several challenges that have to be respected:

- **Challenge 4 – Take into account the low observability and poor controllability:** The operation of conventional computer systems depends fundamentally on the assumption that there is at least one system element equipped with the corresponding capabilities to control or coordinate the activities of all subsystems or other system elements. This becomes feasible due to the high observability and good controllability of conventional systems. In contrast, self-organizing emergent systems lack this/these central control element(s). Because in these systems every element is exposed to incomplete information and limited control of the system, usually no single element has the ability to observe or control the system as a whole. Although these systems thus have no single point of failure or processing bottleneck, a controller will neither be able to observe every (inter)action at the time of occurrence, if ever, nor be able to adapt or influence the behavior, intention, or upcoming action of any system element yielding immediate effects.

- **Challenge 5 – Preserve the basic self-organizing and emergent behavior:** A controller may not limit the underlying system's capabilities to self-organize and to build emergent properties. It may also not decrease the underlying system's inherent properties of scalability, robustness, flexibility, and adaptivity. This implicates that a controller may not execute the role of a central controller prescribing all actions nor become a processing bottleneck or single point of failure. In other words, if the controller crashes, the system and its elements still have to function properly.

- **Challenge 6 – Consider openness and autonomy:** A controller has to be able to cope with previously (i. e. at design time) unknown or unexpected situations. The assessment of the underlying system's behavior has to consider current and past situations, while the behavior or structure of the system has to be adapted autonomously depending on these possibly unforeseen situations. In addition to its own autonomy, a controller may not limit or decrease the underlying system's autonomy. The problem solving decisions still have to be made locally by all participating system elements themselves.

Further Problems and Challenges

Apart from the addressed problems and challenges described above, there exist quite a few more that prevent self-organizing emergent systems from being practically applicable for the effective and efficient usage in various areas. For instance, a crucial research challenge in general is to understand and control self-organizing and emergent phenomena. This requires a fundamental, theoretically profound model of the behavior of complex, biological or technical, systems. Related to the design phase, consistent engineering methodologies have to be provided that focus explicitly on the engineering of macroscopic properties of self-organizing emergent systems. This requires methods to capture and model macroscopic self-* properties and to transform and refine these models into working self-organizing emergent systems. Related to the operation phase, a general problem is the risk of emergent misbehavior [Mog06]. Because already small changes in the behavior of individual system elements may lead to enormous changes in the emergent system behavior (see e. g. [Wil75, BTD+97, CDF+01, Bon02]), approaches are required that provide guarantees on the trustworthiness of self-organizing emergent systems, i. e. their safety, security, reliability, usability, etc. Providing guarantees on the efficiency of these systems can be considered as one aspect of that. More problems and challenges related to self-organizing emergent systems in general can e. g. be found in [MSvdMW04, HMG05, Kep05, Sch05, MBBY06, BBF+08, CGI+09].

1.2 Objectives, Approaches, and Contributions

In the evidence of the grand research challenges and the problems as well as their associated challenges regarding the design and the operation of self-organizing emergent systems described in the previous section, two challenging paradoxes emerge: first, in order to conquer system complexity, we need to create more complex systems in a much more complex way. Second, in order to lower OPEX, we have to use potentially inefficient systems that may increase OPEX in certain situations. Thus, this thesis pursues two main objectives, which are described below along with the approaches taken to achieve them. Additionally, the contributions of this thesis are briefly emphasized.

Objective 1: Simplify the Design of Efficient Self-Organizing Emergent Systems

The first objective of this thesis is to simplify the design of self-organizing emergent systems by developing several artifacts that can be used for the systematic engineering of effective and more efficient solutions. This will particularly reduce the complexity of the design, save development time, and reduce development costs, which consequently reduces TCO.

- **Approach:** In general, in order to achieve an effective coordination, two contrary approaches are possible (cf. [GVCO08]): (1) devising an ad hoc strategy that will solve the specific problem in hand, or (2) observing an existing system that achieves similar results and trying to reverse-engineer its strategy. Whereas it is generally acknowledged that the first approach is only applicable to a limited set of problem domains while being not very sustainable, the second approach, which we will employ in this thesis, is commonly regarded more fruitful. Its successfulness is exemplified by versatile coordination models and mechanisms inspired from physics [MZ04], economics [DZKS06], human societies [Hal06], social science [XSY+05], or biology [BCD+06], for instance. Such paradigms very often impressively demonstrated over thousands of years, how a global functionality emerges from local processes and which local behaviors are required to achieve certain global properties in an autonomous manner.

  In more detail, in this thesis we investigate the general principles behind decentralized coordination by infochemicals in biological systems [DS88] and adopt them into the computational world. Infochemical-based coordination (IBC) is the most universally employed communication and coordination model between organisms in biology (cf. [Lew84]), and hence provides a plethora of inspiring examples that can be adopted as coordination mechanisms for self-organizing emergent solutions to artificial problems in various application domains. Pheromone-based coordination, as used during foraging in ant colonies, is probably the best-known example for a coordination mechanism based on the principles of IBC. However, pheromones constitute only one single type of infochemicals. The principles of IBC in general allow for the combination of different types of infochemicals within one and the same coordination mechanism as well as a combination of quantitative and qualitative stigmergy, which consequently allows for the design of better adaptable and efficient solutions compared to existing coordination approaches.

  In this thesis, we capture these general principles in a model for decentralized coordination, which is as a direct consequence versatile and coherent, and thus cope with *Challenge 1*. Based on the adopted coordination model and inspired from the plethora of examples a vast amount of new and efficient coordination mechanisms can be specified, which are naturally based on and guided by the expressiveness of the coordination model, which copes with *Challenge 2*. Furthermore, we capture the principles of IBC not only in a formal model, but

also in a design pattern, as well as develop design guidelines, how to use this pattern technically, and thus cope with *Challenge 3*. This allows engineers to specify new coordination mechanisms themselves, but does not force the engineers to be biological experts at the same time.

- **Contributions:** Due to the described approach, this thesis presents the following artifacts that can be used for the simplified design of more efficient self-organizing emergent systems:
  - A decentralized coordination model for self-organizing emergent systems, which is based on the general principles behind IBC in biology and hence called digital infochemical coordination (DIC). The model facilitates the efficient coordination of homogeneous as well as heterogeneous system elements. Furthermore, it provides the basis for the identification as well as specification of multiple new coordination mechanisms.
  - A design pattern that describes the DIC model in a systematic way familiar to software engineers. The pattern hides the inherent complexity of the designed system and makes meaningful abstractions from the biological principles. Both simplifies the instantiation of DIC in terms of new coordination mechanisms and hence promotes the application of DIC to a wider problem spectrum.
  - Design guidelines that support the instantiation and usage of DIC for the purpose of specifying new coordination mechanisms. This helps software engineers in identifying the required types of digital infochemicals that have to be combined in a coordination mechanism for a problem in hand.
  - A decentralized coordination mechanism based on the DIC model, which takes its inspiration from the pollination of flowers by honey bees. This so-called pollination-inspired coordination (PIC) mechanism demonstrates the many beneficial capabilities of the developed approach in a concrete application domain.
  - A Simulator for EfficieNt Self-Organizing Emergent Systems (SENSES) that facilitates the identification, simulation, and improvement of decentralized coordination mechanisms regarding a problem in hand. SENSES can be integrated into existing engineering methodologies and cuts down the design time.

Objective 2: Facilitate an Efficient Operation of Self-Organizing Emergent Systems

The second objective of this thesis is to facilitate an efficient operation of self-organizing emergent systems by developing an approach that autonomously adapts the local behavior of self-organizing agents in order to improve the efficiency of the global solution in certain situations.

- **Approach:** To overcome the limits of systems in guaranteeing an output of a desired quality for dynamic problems, a well known approach – originating mainly from control theory – is the integration of a closed feedback control loop on top of the basic system. Instantiations of this general concept can be found in several areas, such as Autonomic Computing (see [KC03]), Organic Computing (see [RMB+06]), or multi-agent systems (see [SLT08]), for instance.

  In contrast to existing instantiations, the basic approach pursued in this thesis is to only provide advice to a self-organizing emergent system instead of controlling it. A dedicated system element, consequently called advisor, collects the local history of the system elements and, based on the aggregated global history, autonomously detects recurring task patterns the system had and potentially will have to fulfill, but which are currently solved far from optimal. Based on a global optimization function the advisor calculates the optimal solution for these task patterns and, if necessary, provides the elements with advice in form of exception rules that the elements can add to their own problem solving behavior. As a result, all problem solving decisions are still locally made by the agents, in order to cope with *Challenge 6*. In contrast to a central control (and therefore the nearly total loss of autonomy of the elements), the advisor only communicates with the elements when interaction is possible. The advice can be ignored by the elements (and essentially will be ignored, if it is not useful anymore) and the elements will continue to work even if the advisor fails, which copes with *Challenge 5*. This frees the system from a single point of failure. Furthermore, instead of trying to figure out how the system has to work and what has to be optimized a priori, a learning mechanism based on actual data helps the system elements to perform better and to develop more efficient and coordinated solutions at runtime, in order to cope with *Challenge 4*, which requires no intervention by a human user. By this advisor approach, we obtain some of the benefits of a central control but avoid its associated problems.

- **Contributions:** Due to the described approach, this thesis presents the following artifacts that facilitate an efficient operation of self-organizing systems:
  - A general architecture of an advisor able to improve the efficiency of self-organizing emergent systems solving dynamic problems with recurring tasks. This so-called Efficiency Improvement Advisor (EIA) realizes an unobtrusive feedback and learning mechanism that is independent of the coordination model or mechanism used by the underlying self-organizing emergent system as well as the application domain in hand.
  - A multi-level classification of exception rules that can be used by an EIA for adapting the local behavior of system elements in order to improve the global self-organizing emergent solution.
  - A customized instantiation of the EIA approach for the use in a concrete application domain, demonstrating the gained efficiency improvements.

In order to prove and illustrate the artifacts developed in this thesis, we apply them exemplary to a concrete application domain. In the research area of self-managing systems, the application focus is usually on technology and communication infrastructures, such as huge server farms or data centers. However, more and more researchers become aware of the fact that in order to exploit the full potential of OPEX savings the entire product chain has to be regarded. This fact extends the application focus in this research area to facility, production, and mobility infrastructures as well. Because self-organizing emergent systems in general are an eligible candidate for solutions to dynamic optimization problems that can be found in the latter areas, e. g. resource[2] allocation, load balancing, spatial distribution, dynamic clustering, group formation, or re-organization problems, we apply the developed artifacts to the domain of pickup and delivery problems.

Roughly spoken, a Pickup and Delivery Problem (PDP) concerns the service of a set of customers in a given time period by a set of vehicles, which are located in one (or more) depots and perform their movements by using an appropriate road network. A solution of a PDP calls for the determination of a set of routes, each performed by a single vehicle that starts and ends at its own depot, such that all requirements of the customers are fulfilled, all the operational constraints are satisfied, and one or more global optimization objectives are reached (cf. [TV02]). Practically, the PDP is an omni-present problem that appears in various areas of logistics, such as courier services, manufacturing control, aircraft sharing, dial-a-ride transportation, container terminals, delivery of heating oil, taxi cab services, emergency vehicle dispatching, or even patient transportation in hospitals, to name only a few. Apparently, in all of these areas, efficiency plays a major role. For instance, the reduction of logistics costs, which represent to a great extent OPEX, has the highest priority for the world's leading food, beverage, and consumer products companies [IBM08]. Moreover, high logistics costs for instance caused by inefficient vehicle routes are also responsible for a tremendous energy consumption as well as a poor carbon footprint, which make energy-saving, efficient solutions also highly valuable for nature.

Solving PDPs efficiently by hand is impossible as soon as the number of vehicles, requests, customers, packages, constraints, or restrictions exceed a certain, rather small limit. Solving PDPs efficiently by conventional solution methods, such as exact algorithms, heuristics, or meta-heuristics (see e. g. [BCGL07, PDH08a, PDH08b]), is impossible as well, as soon as the problem size and in particular the dynamics exceed a certain limit. This handicap calls for self-organizing emergent solutions. However, their complex and costly design as well as their uncertain runtime efficiency compared to conventional solutions currently prevents them from being applied to such problem classes in practice.

[2] A resource can be a task, power, bandwidth, space, (CPU) time, a device, a machine, etc.

1.3 Outline

The structure of this thesis is illustrated in Figure 1.1. Even though a straightforward reading is recommended, the arrows indicate possible reading sequences. Part I deals with a concise introduction to and basics of self-organizing emergent systems. Experienced readers that are familiar with these topics may skip Chapter 2 or at least some parts of it. Readers then have the choice of reading first Part II that deals with the design phase of self-organizing emergent systems, i. e. Chapters 3 and 4, or reading first Part III that deals with their operation phase, i. e. Chapters 5 and 6. Part IV then builds upon the content of the prior parts and deals with the application of the developed concepts to a concrete problem domain in Chapter 7 and its experimental evaluation in Chapter 8. Finally, Chapter 9 presents the conclusions of this thesis and an outlook on future work. In more detail, the individual chapters have the following content:

Chapter 1 – Introduction

This chapter introduces the class of computer systems considered in this thesis and identifies problems and challenges related to their design and operation. It defines the thesis' objectives, sketches the taken approaches to achieve them, and briefly emphasizes the contributions of this thesis. At the end an overview of the chapters of this thesis is provided and publications are listed, in which parts of this thesis have been previously published.

Chapter 2 – Basics

This chapter presents the necessary background knowledge for this thesis. It discusses the basic terms and concepts used across the remaining chapters, more specifically self-management, self-adaptation, self-organization, as well as multi-agent systems, and elaborates on their definitions. Furthermore, this chapter surveys research areas related to this thesis.

Chapter 3 – Designing Self-Organizing Emergent Systems

This chapter presents the background for and state of the art in designing self-organizing emergent systems. In more detail, it deals with the concept of coordination – as a key issue for the design phase – and provides a comprehensive survey on existing coordination models and languages for computer systems in general and multi-agent systems in particular. A special focus thereby is on models and mechanisms for decentralized coordination. Finally, this chapter examines the design process of self-organizing emergent systems in the context of existing engineering methodologies.

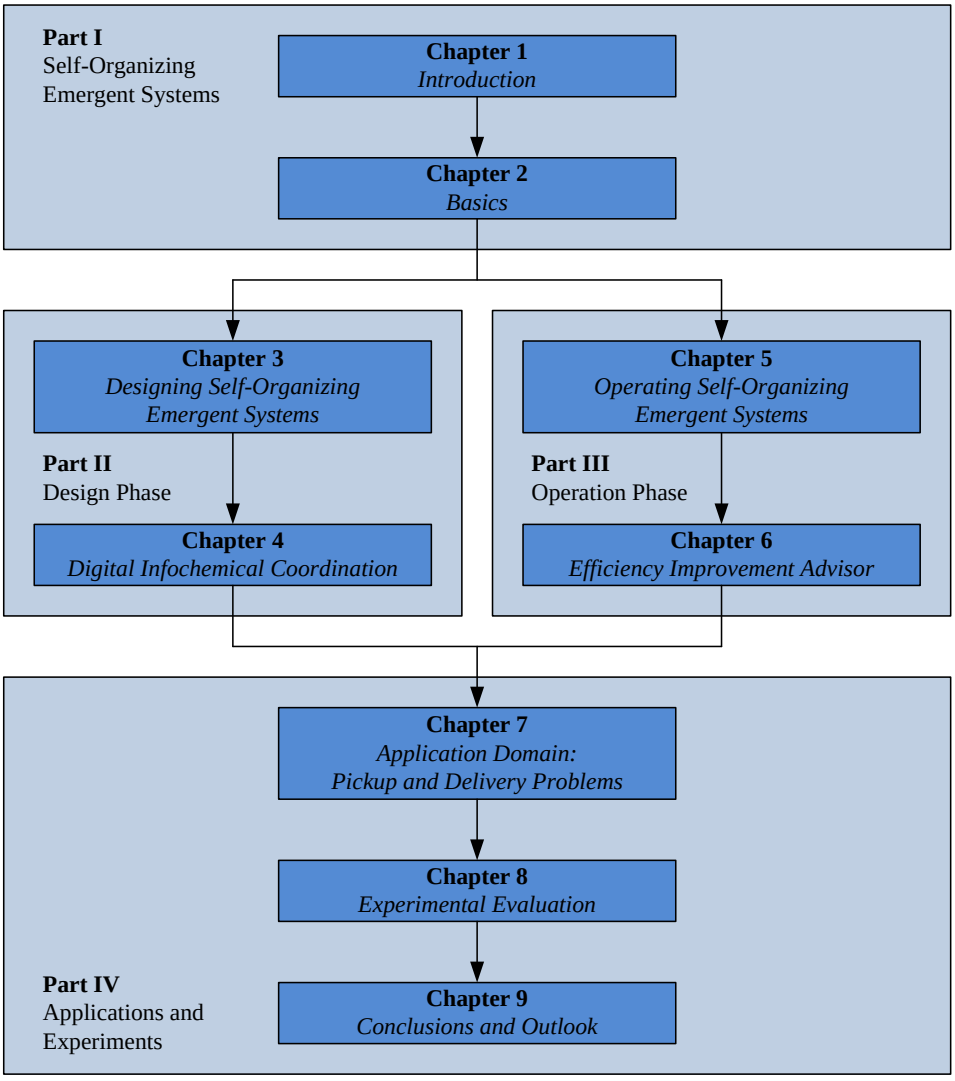


Figure 1.1: Thesis structure

Chapter 4 – Infochemical-based Coordination

This chapter starts with a concise introduction into IBC in nature and presents the adaptation of the general principles behind IBC for the computational world by a formal coordination model. The chapter subsequently describes the coordination model as a design pattern for self-organizing emergent systems and provides corresponding design guidelines. Furthermore, it exemplary instantiates the coordination model by the existing pheromone-based coordination mechanism.

Chapter 5 – Operating Self-Organizing Emergent Systems

This chapter presents the background for and state of the art in operating self-organizing emergent systems. In more detail, it lists reasons for the runtime inefficiencies of self-organizing emergent systems and provides necessary foundations on control theory as well as different types of control systems to compensate these inefficiencies in principle. This chapter furthermore describes several reference models for the control theory-based adaptation of computer systems in general and surveys existing approaches and architectures for adapting self-organizing emergent systems in particular.

Chapter 6 – Efficiency Improvement Advisor

This chapter presents the general architecture of the Efficiency Improvement Advisor for the adaptation of self-organizing emergent systems during their operation. This chapter first introduces the terminology and premises used for this approach and then describes the formal foundation of this architecture in detail. Finally, this chapter considers several aspects with regard to the realization of this approach.

Chapter 7 – Application Domain: Pickup and Delivery Problems

This chapter starts with a concise introduction into the application domain of PDPs, to which we apply the concepts developed in Chapter 4 and Chapter 6 to. It then presents PIC as a biologically-inspired coordination mechanism based on IBC that can be used for a more efficient self-organizing emergent solution to this class of dynamic optimization problems. Likewise, this chapter presents an instantiation of the EIA architecture customized to PIC for this application domain as well.

Chapter 8 – Experimental Evaluation

This chapter experimentally evaluates the concepts developed in Chapter 4 and 6 based on the instantiations presented in Chapter 7. This chapter first deals with the developed simulation tool used to execute the experiments. It then presents two case studies from the application domain of PDPs and finally explains the experiment execution and analyzes the experimental results.

Chapter 9 - Conclusions and Outlook

The last chapter summarizes the contributions presented in this thesis and discusses their limitations and implications. Based on these results, the thesis ends with an outlook on future work.

1.4 Publications

Parts of this thesis have been previously published in the following peer-reviewed publications:

1. Holger Kasinger and Bernhard Bauer. Towards a Model-Driven Software Engineering Methodology for Organic Computing Systems. In *Proceedings of the IASTED International Conference on Computational Intelligence (CI 2005)*, pages 141–146. ACTA Press, 2005.

2. Holger Kasinger and Bernhard Bauer. Combining Multi-Agent-System Methodologies for Organic Computing Systems. In *Proceedings of the 3rd International Workshop on Self-Adaptable and Autonomic Computing Systems (SAACS 2005), part of DEXA Workshop 2005*, pages 160–164. IEEE Computer Society, 2005.

3. Bernhard Bauer and Holger Kasinger. AOSE and Organic Computing - How Can They Benefit from Each Other?. In *Proceedings of the 7th International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS2005), Revised Selected Papers*, volume 3529 of *Lecture Notes in Computer Science*, pages 154–167, 2006. Springer.

4. Holger Kasinger and Bernhard Bauer. The utility of pollination for autonomic computing. In Yi Pan, Franz Rammig, Hartmut Schmeck, and Mauricio Solar, editors, *Biologically Inspired Cooperative Computing*, volume 216 of *IFIP International Federation for Information Processing*, pages 55–64. Springer Boston, 2006.

5. Holger Kasinger and Bernhard Bauer. Pollination – a biologically inspired paradigm for self-managing systems. *Journal of International Transactions on Systems Science and Applications*, 2(2):147–156, September 2006.

6. Holger Kasinger and Bernhard Bauer. Beyond Swarm Intelligence: Building Self-Managing Systems Based on Pollination. In *Informatik 2006 - Informatik für Menschen, Band 1, Beiträge der 36. Jahrestagung der Gesellschaft für Informatik e.V. (GI)*, volume 93 of *Lecture Notes in Informatics*, pages 169–176, 2006. GI.

7. Holger Kasinger, Bernhard Bauer, Henning Sanneck and Christoph Schmelz: A Management Automation Framework for Mobile Networks. In WWRF Meeting #17 - Special Interest Group 3, 2006.

8. Holger Kasinger, Jörg Denzinger, and Bernhard Bauer. Digital semiochemical coordination. *Communications of SIWN*, 4:133–139, June 2008.
***** Best Student Paper Award SIWN 2008 Congress *****

9. Holger Kasinger, Bernhard Bauer, and Jörg Denzinger. The meaning of semiochemicals to the design of self-organizing systems. In *Proceedings of the 2nd IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2008)*, pages 139–148. IEEE Computer Society, 2008.

10. Holger Kasinger, Jörg Denzinger, and Bernhard Bauer. Decentralized coordination of homogeneous and heterogeneous agents by digital infochemicals. In *Proceedings of the 24th Annual ACM Symposium on Applied Computing (SAC 2009)*, pages 1223–1224. ACM Press, 2009.

11. Holger Kasinger, Bernhard Bauer, and Jörg Denzinger. Design pattern for self-organizing emergent systems based on digital infochemicals. In *Proceedings of the 6th IEEE International Conference and Workshop on Engineering of Autonomic and Autonomous Systems (EASe 2009)*, pages 45–55. IEEE Computer Society, 2009.

12. Michael Rambold, Holger Kasinger, Florian Lautenbacher, and Bernhard Bauer. Towards Autonomic Service Discovery – A Survey and Comparison. In *Proceedings of the 2009 IEEE International Conference on Services Computing (SCC 2009)*, pages 192–201. IEEE Computer Society, 2009.

13. Jan-Philipp Steghöfer, Jörg Denzinger, Holger Kasinger and Bernhard Bauer. Improving the Efficiency of Self-Organizing Emergent Systems by an Advisor. In *Proceedings of the 7th IEEE Conference and Workshop on Engineering of Autonomic and Autonomous Systems (EASe 2010)*, pages 63–72. IEEE Computer Society, 2010.

14. Holger Kasinger, Bernhard Bauer, Jörg Denzinger and Tom Holvoet. Adapting Environment-Mediated Self-Organizing Emergent Systems by Exception Rules. In *Proceedings of the 2nd International Workshop on Self-Organizing Architectures (SOAR 2010)*, pages 35–42. ACM, 2010.

15. Florian Dötsch, Jörg Denzinger, Holger Kasinger and Bernhard Bauer. Decentralized Real-time Control of Water Distribution Networks Using Self-organizing Multi-Agent Systems. In *Proceedings of the 4th IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2010)*, pages 223–232. IEEE, 2010.

Chapter 2

Basics

The increasing complexity of computer systems and applications as well as the need for autonomy and continuous availability has led different communities, e. g. software engineering (SE), distributed systems, or system management, to look for inspiration in diverse fields, such as complex systems, artificial intelligence, sociology, and biology, to find new ways of designing and managing networks, systems, and services. In this endeavor, three promising, interrelated facets of this paradigm shift have emerged: self-management, self-adaptation, and self-organization.

Although every single one of them has its own roots, their interrelation, combination, and convergence is highly valued for tackling the challenges of complexity. This can be deduced, for instance, from the workshop and conference series being held in recent years, aiming at a cross-pollination between these different facets: the SASO (Self-Adaptive and Self-Organizing Systems) conference series [SMFJZ07, BRB08, MSM+09] emerged from the workshop series ESOA (Engineering Self-Organizing Applications) [SKRZ04, BSKN05, BSHZ06, BHJY07], SelfMan (Self-Managed Networks, Systems and Services) [MFSG05, KMF06], Self-Star (Self-* Properties in Complex Information Systems) [BJM+05], and IWSAS (International Workshop on Self-Adaptive Software) [RSL01, LRS03b]. Likewise, the SACC (Self-organization and Adaptation of Computing and Communications) conference series [Tia08, Tia09] evolved from the SOAS (Self-Organization and Autonomous Systems in Computing and Communications) conference series [CUBT05, Tia06, Tia07]. A workshop series that aims to combine principles from self-adaptation and self-management is SEAMS (Software Engineering for Adaptive and Self-Managing Systems) [CdF+06, CdF+07, CdG+08, CdG+09]. A workshop series that by contrast aims to combine principles from self-adaptation and self-organization is SOAR (Self-Organizing Architectures) [WMdLA10, WMAS10].

In the subsequent three sections of this chapter we survey these three interrelated terms and concepts in more detail. Section 2.1 focuses on self-management, Section 2.2 on self-adaptation, and Section 2.3 on self-organization along with emergence. This provides a clarifying understanding of these terms, necessary to classify mechanisms, techniques, and approaches mentioned and developed in this thesis, and results in the definition of self-organizing emergent systems as understood in this thesis. Section 2.4 presents the basics of multi-agent systems as an eligible technology to model and realize self-organizing emergent systems. In Section 2.5 we survey industrial and academic research areas related to the tackled problems

and challenges of this thesis, first and foremost Autonomic Computing and Organic Computing. Finally, Section 2.6 concludes this chapter.

2.1 Self-Management

The term self-management rather describes a vision than a elaborated technical concept. The vision of self-management is *"to free system administrators from the details of system operation and maintenance and to provide users with a machine that runs at peak performance 24/7"* [KC03]. The desire to move control from the human to the system is rational, because administrators have a limited response rate, often make mistakes, are expensive to maintain, and difficult to train. Although the idea of self-managing systems is actually very old and dates back to the 1960s (see e. g. [Wie65, vB69]), they have recently come into the limelight due to IBM's Autonomic Computing Initiative [Hor01] (see later Subsection 2.5.1). This is why self managing systems are also often referred to as autonomic systems respectively why self-management is often referred to as autonomicity (cf. [HS06, THRR06]).

At this point, one has to be aware of the difference made by the Autonomic Computing community between autonomy and autonomicity. Generally, a system is considered as autonomous, if it can achieve its goals without human intervention (cf. [THRR06]). Thus, *autonomy* (in the sense of self-governance or self-direction) is seen as the delegation of responsibility to the system to meet the defined *goals* of the system, i. e. an automation of responsibility including some decision making for the success of tasks. For instance, the goal of a system (potentially implemented as multi-agent system) may be to find the best schedule for processing certain jobs on machines in a manufacturing environment. Therefore, the system may have the autonomy to negotiate between certain time windows, price ranges, budgets, priorities, and penalties. However, with autonomy alone, absent autonomicity, the performance of the system might degrade in unforeseen situations or the system might not be able to recover from faults. This would not fall under this specific delegated task of the agents, i. e. their autonomy. Thus, in contrast, *autonomicity* is seen as the self-management of the system, i. e. an automation of responsibility including some decision making for the successful *operation* of the system. It may be considered as specialized form of autonomy, that is the autonomy is specifically to manage the system (cf. [SGHO06]). In this sense, our approach presented in Chapter 4 will promote the autonomy of a computer system, whereas our approach presented in Chapter 6 will promote a system's autonomicity.

Due to the proximity to Autonomic Computing, the properties of a self-managing, or autonomic, system can be summarized by four objectives and four attributes (see Figure 2.1). Essentially, the objectives represent broad system requirements, while the attributes identify basic implementation mechanisms.

Self-configuration enables the system to adapt to unpredictable conditions by automatically changing its configuration, such as adding or removing new elements, or installing software changes without disrupting service. *Self-optimization* enables

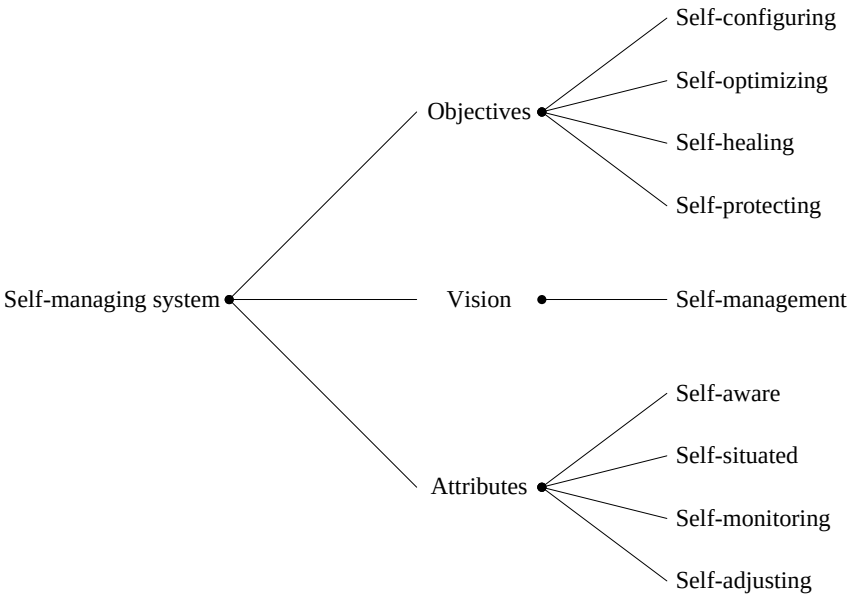


Figure 2.1: Self-managing system properties tree (adapted from [Ste05])

the system to continuously tune itself, either proactively to improve on existing processes or reactively in response to environmental changes. *Self-healing* prevents and recovers from failures by automatically discovering, diagnosing, circumventing, and recovering from issues that might cause service disruptions. *Self-protection* detects, identifies, and defends against viruses, unauthorized access, or denial-of-service attacks. To achieve these objectives, a self-managing system ought to be (cf. [Hor01]): *self-aware*, i. e. to have knowledge of its elements, status, capacity, etc., but also of the context of its activity and those of other elements within the infrastructure; *self-situated* (environment-aware), i. e. to sense and analyze environmental conditions, which includes both to proactively take the pulse of other elements and looking for ways to improve the system's functions as well as to notice change and understand the implications of that change, which means *self-monitoring*; *self-adjusting*, i. e. to plan for and affect change by altering its own state and effecting changes in other elements. However, some of these attributes only apply to centralized systems.

From an abstract point of view, two different categories of approaches can be distinguished to enable self-management (cf. [WHE+08]): *exogenous self-management*, which works in a top-down style considering an individual system, and *endogenous self-management*, which works in a bottom-up style considering cooperative systems. Individual self-managing systems assess their own behavior and change it when the assessment indicates that they are not accomplishing what they were intended to do, or when better functionality or performance is possible. For these purposes, an additional system element (or subsystem) is added on top of the system. The subsystem realizes a control loop, i. e. it monitors and analyzes the underlying system at runtime and adapts the structure or behavior of the system based on an explicit

internal representation of the system and high-level objectives specified by humans (for a more detailed description of control loops see later Section 5.2). Exogenous self-management most closely corresponds to the understanding of self-adaptation (see Section 2.2). In contrast, bottom-up self-managing systems are composed of a large number of elements that interact locally according to simple rules. In these systems, the system elements adapt their structure or behavior to changing requirements themselves and hence cooperatively realize self-management. Consequently, endogenous self-management most closely corresponds to the understanding of self-organization [JBL06] (see Section 2.3).

To summarize, self-management offers a vision for the development and evolution of software, brings new levels of automation, autonomy, and autonomicity to systems, while simultaneously hides their complexity, which at large reduces costs. The two approaches for self-management in form of individual and cooperative systems respectively self-adaptive and self-organizing systems are often considered as two extreme poles of a self-management spectrum. In practice, the line between both is rather blurred, and compromises will often lead to an engineering approach incorporating representatives from these two extreme poles (cf. [WHE+08]). Even this thesis uses and develops concepts from both sides of the spectrum, as we use concepts of self-organization to design efficient self-managing systems but also use concepts of self-adaptation to operate self-organizing systems efficiently.

2.2 Self-Adaptation

The actual motivation for self-adaptive software and systems arose from society's increasing dependence on software-intensive systems and the real risks, costs, and inconvenience that their downtime presents (cf. [OMT08]). More and more systems are required to work continuously, and to do so in environments where users' requirements or system resources may change frequently. This change makes the continuous availability a critical requirement for certain classes of software systems. Consequently, this fact brakes with the implicit assumption in software engineering that systems are designed as well as maintained offline. To address this new kind of software systems, engineers need cost-effective techniques and mechanisms to build reliable systems that adapt their own behavior dynamically.

One of the earliest definitions of self-adaptive systems respectively software was given by Oreizy et al. [OGT+99] in 1999: *"Self-adaptive software modifies its own behavior in response to changes in its operating environment. By operating environment, we mean anything observable by the software system, such as end-user input, external hardware devices and sensors, or program instrumentation. . . . [The system] observes its own behavior and analyzes these observations to determine appropriate adaptations."* A similar but more detailed definition was given by Laddaga et al. [LRS03a] some years later: a *"[s]elf-adaptive software evaluates its own behavior and changes behavior when the evaluation indicates that it is not accomplishing what the software is intended to do, or when better functionality or performance is*

possible. ... This implies that the software has multiple ways of accomplishing its purpose, and has enough knowledge of its construction to make effective changes at runtime. Such software should include functionality for evaluating its behavior and performance, as well as the ability to replan and reconfigure its operations in order to improve its operation." Both definitions characterize the demand for systems whose environment changes at a rate that necessitates the system to adapt[1] at runtime.

However, at that early date, self-adaptation and self-management had been two separate research fields and hence none of the definitions makes a clear declaration on the involvement of humans in the adaptation process. The convergence of these two research fields is expressed by a recent definition, which resulted from a seminar on software engineering for self-adaptive systems [CGI+09]: *"self-adaptive systems can configure and reconfigure themselves, augment their functionality, continually optimize themselves, protect themselves, and recover themselves, while keeping most of their complexity hidden from the user and administrator."* Similarly, the vision of self-adaptation and self-management are closely related, too: *"The vision of self-adaptation is that software systems can autonomously adapt themselves to context changes and handle changes of the requirements on their own"*[2] [GGH08].

In general, there are two different approaches to implementing self-adaptation (cf. [MSKC04]): parameter adaptation and compositional adaptation. *Parameter adaptation* modifies program variables that determine the behavior of the system. It can tune parameters or direct a system to use a different existing strategy. However, it cannot adopt new strategies and does not allow new algorithms and elements to be added to an application after the original design and implementation. By contrast, *compositional adaptation* exchanges algorithmic or structural system elements with others that improve an application's fit to its current environment. Thus, an application can adopt new algorithms for addressing concerns that were unforeseen during development. This flexibility supports more than simple tuning of program variables or strategy selection, it enables dynamic recomposition of the software during runtime.

Compositional adaptation in most earlier applications has been implemented in a fairly ad hoc fashion. Mechanisms that support self-adaptation have been around for a long time in the form of programming language features, e. g. Java exceptions or runtime assertion checking, and algorithms, e. g. network protocols or self-stabilizing algorithms such as timeouts for Remote Procedure Calls (RPC). The code that deals with self-adaptation was typically tightly integrated with the application itself and wired at the code level. Such "internal" mechanisms suffer from the problem that localized error handling may not be able to determine the true source of the problem, and consequently are not able to determine the required remedial action. Moreover, while they can only trap an error at the moment of detection, they are not well-suited to recognizing "softer" system anomalies, such as gradual degradation

[1] In computer science literature, the terms 'self-adaptive' and 'adaptive' are very often used synonymously.

[2] The term 'context' has a very general meaning in this vision.

of performance, or patterns of unreliability. Because internal mechanisms are so intertwined with the normal code of the system, they complicate both the application and adaptation code and make the change or reuse of adaptation strategies impossible (cf. [GS02]).

McKinley et al. [MSKC04] propose three key technologies for compositional adaptation, which can be used by software engineers to construct self-adaptive systems in a systematic and principled – as opposed to ad hoc – fashion : separation of concerns, computational reflection, and component-based design. *Separation of concerns*, which is also an important principle in mainstream software engineering, enables the separate development of an application's functional behavior, i. e. its business logic, and the code for crosscutting concerns, such as quality of service (QoS), energy consumption, fault tolerance, and security. *Computational reflection* refers to an applications's ability to reason about, and possibly alter, its own behavior. It comprises two activities: introspection, i. e. to let an application observe its own behavior, and intercession, i. e. to let a system or application act on these observations and modify its own behavior. The third technology for compositional adaptation is *component-based design*. It supports two types of composition: in static composition, a developer can combine several components at compile time to produce an application, whereas in dynamic composition, the developer can add, remove, or reconfigure components within an application at runtime (which requires late binding).

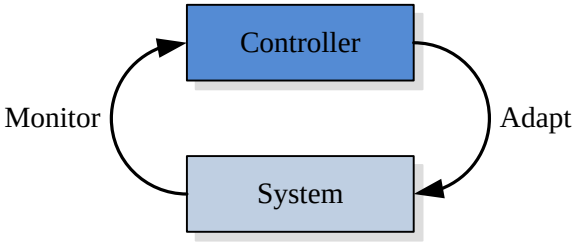


Figure 2.2: Abstract closed control loop

Consequently, there is a consensus in the research community that general solutions, which separate and externalize adaptation mechanisms and control from the application, are essential to achieve real self-adaptation (cf. [GCH+04]). "Externalized" adaptation (exogenous self-management) supports a kind of closed loop control system paradigm as depicted in Figure 2.2 (see also later Section 5.2 for more details). In this paradigm, the system behavior is monitored and analyzed by externalized mechanisms encapsulated in a controller outside the running system. These mechanisms are responsible for (1) determining when a system's behavior is within the envelope of acceptable system parameters, and (2) when it falls outside of those limits, adapting the system. To accomplish these tasks, the mechanisms usually maintain one or more system models, which provide an abstract, global view of the running system, and support reasoning about system problems and repairs.

A model can thereby be any abstract representation of the system, such as architecture models, performance models, reliability models, etc. (see e. g. [KS99, GS02]). Our approach for the efficiency improvement of self-organizing emergent systems at runtime (see Chapter 6) is likewise based on the principle of externalized adaptation.

To summarize, self-adaptation is understood as exogenous self-management considering an individual system that is enhanced by an additional system element implementing a control loop to evaluate and change the system's behavior when better functionality or performance is possible. Although the focus in the above paragraphs has been primarily on self-adaptation on the application level, there also exist many approaches for self-adaptive middlewares (see e. g. [MSKC04, GEL+06, Tru06]) and self-adaptive operating systems (see e. g. [SS97]). However, for the purpose of this thesis, we will maintain the focus on the application level.

2.3 Self-Organization and Emergence

Self-management through self-organization, i. e. endogenous self-management, is motivated by providing a variety of systems such as wired or wireless networks, P2P systems, the computational Grid, as well as distributed and embedded systems and applications with self-managing capabilities (cf. [JBL06]). But in contrast to self-adaptation, to this day there exists no common understanding and generally accepted definition of *self-organization* that satisfies everyone, although or just because this concept is not a product of modern times but has already a long history[3].

2.3.1 History of Self-Organization

In 1637, Descartes was probably the first capturing the essence of self-organization by noting a spontaneous, dynamically-produced organization [Des85]:

> *[Consider] what would happen in a new world, if God were now to create somewhere in the imaginary spaces matter sufficient to compose one, and were to agitate variously and confusedly the different parts of this matter, so that there resulted a chaos as disordered as the poets ever feigned, and after that did* ***nothing more than lend his ordinary concurrence to nature, and allow her to act in accordance with the laws which He had established*** *. . . . I showed how the greatest part of the matter of this chaos must, in accordance with these laws, dispose and* ***arrange itself*** *in such a way as to present the appearance of heavens; how in the meantime some of its parts must compose an earth and some planets and comets, and others a sun and fixed stars.*

This phenomenon was called self-organization not until the years after World War II, primarily in research connected with cybernetics and computing machinery [YC60, vFJ62]. The first appearance of the term at all seems to be in a paper

[3] The following historical sketch of self-organization is partly based on the work in [Sha01].

by Ashby [Ash47] in 1947. He gave a pretty clear explanation of what he meant by 'organization': to paraphrase, the organization of a system is the functional dependence of its future state on its present state and its external inputs, if any. That is, if the state space is S and the input space is I, the organization of the system is the function $f : S \times I \to S$ which gives the new state (see also [Ash60]). Ashby understood a system to be self-organizing if it changed its own organization, rather than being rewired by an external system.

The main research domains, in which the phenomenon of self-organization was studied for the next decades, were physics, computer science, and systems theory. In the physical sciences, self-organization was extensively applied from the 1970s onwards to pattern formation and spontaneous symmetry breaking [NP77] as well as to cooperative phenomena [Hak77]. Within computer science, the primary application areas have been learning [Sel59, YC60], especially unsupervised learning [HS99a] and memory [Koh84, Koh01], to adaptation [FLPW86, Hol92], and to 'emergent' or distributed computation [For90, Res94, Cru94a, CM95]. In the 1980s, self-organization became also one of the ideas, models, and techniques bundled together as the 'sciences of complexity' [Pag88]. This bundle has been successful at getting itself adopted by researchers in essentially every science, so the idea of self-organization is now used in a huge range of disciplines (cf. [Sha01]).

Thus, various definitions, formalizations, and examples of self-organization can be found in diverse scientific disciplines today, e. g. in ecology [Wal90, CDF+01, FCG06], economics [Sch78, Kru96], mathematics [Len64], complexity [Sch97], information theory [Sha01, SS03], cybernetics [vF60, Ash62, HJ01, Hey03], synergetics [Hak06], biology [BTD+97, BDT99], and not least computer science [PB01, WSDG01, HG03, GH03, DH04, SGK05, MMTZ06, MWJ+07, CMMS+07].

2.3.2 Shapes of Self-Organization

The abundance of appearances of self-organization indicates that the understanding and definition of self-organization strongly depends on the discipline it appears in. To illustrate this, we will focus on three different appearances of self-organization observable in the natural world. These examples essentially enhance the fact that there exist multiple shapes of self-organization, which are opposed to a general definition.

2.3.2.1 Stigmergy

By studying the social behavior of swarms, in more detail insects (termites), Grassé [Gra59] proposed in 1959 the theory of *stigmergy*. This theory can be summarized in short as "the work excites the workers". The consequence of this theory is that direct interactions and regulation by a central control are not necessary to coordinate a group. To the contrary, coordination and regulation tasks are realized on the basis of information deposited into the environment, without central control. In the case of ants, termites, and honey bees, stigmergy is ensured by depositing a chemical

substance (stimulus) in the environment, called *pheromone*. Thus, self-organization results from the behavior or response (of the insects) arising from inside the system, so the elements of a swarm are themselves at the origin of the re-organization of the whole system. The self-organizing behavior is however not limited to insects only. Other collective behaviors of animals referred to as being self-organizing are flocks of birds and schools of fish, for instance (see [CDF+01]). By following simple rules, such as getting close to a similar bird (or fish) but not too much or getting away from dissimilar birds (or fishes), they are e. g. able to collectively avoid predators. Such a self-organizing behavior is commonly also referred to as swarm intelligence [GGT07]. According to Bonabeau et al. [BDT99], self-organization in systems based on swarm intelligence relies on four basic ingredients:

- **Positive feedback:** simple behavioral "rules of thumb" promote the creation of structures (amplification). For example, in ant foraging, recruitment to a food source is a positive feedback that relies on trail laying by pheromones and trail following by other ants. Reinforcement is another positive feedback.
- **Negative feedback:** counterbalance to positive feedback that helps to stabilize the collective pattern. In the example of ant foraging, the limited number of foragers, saturation, food source exhaustion, and crowding at food source are forms of negative feedback that hamper a refreshment of the pheromone trail.
- **Amplification of fluctuations:** Randomness plays a crucial role, not only for the emergence of structures but also for the discovery of new solutions. For instance, ant foragers may get lost because they follow trails with some error. However, thereby they can find new, unexploited food sources and recruit nestmates to these sources.
- **Multiple interactions:** Individuals should be able to make use of the results of their own activities as well as of other's activities. For instance, ant trail networks can self-organize and be used collectively if individuals use other's pheromones.

In general, there exist four different varieties of stigmergy (see Table 2.1 with examples) that can be distinguished along two orthogonal dimensions regarding the stimulus respectively the response (cf. [BDT99, TB99, CDF+01, And02]). The first dimension considers whether the stimuli represent special markers, e. g. pheromones, that individuals deposit in the environment ("marker-based stigmergy") or whether only domain-specific elements, e. g. dead ant bodies, are used for the stigmergic effect ("sematectonic stigmergy"). The second dimension considers whether the stimuli are a single scalar analogous to a potential field ("quantitative stigmergy") or whether they form a set of discrete options ("qualitative stigmergy"). In the case of quantitative stigmergy, the stimulus varies in a quantitative manner, e. g. the local pheromone concentration, altering the probability of eliciting the same

response from other individuals. In the case of qualitative stigmergy, the stimuli differ from each other qualitatively, e. g. different types of pheromones, but not in their quantity and as a result may elicit different responses, see e. g. nest construction in Polistes wasps (see [BDT99, TB99, CDF+01]). This latter variety of stigmergy usually does not explicitly require positive feedback, even though there may be cases in which positive feedback is not required for certain quantitative examples, too (see [And02]).

	Marker-based Stimuli inserted into the domain	**Sematectonic** Domain stimuli only
Quantitative Scalar quantities	e. g. gradient following in a single pheromone field	e. g. ant cemetery clustering
Qualitative Symbolic differentiations	e. g. decisions based on combinations of pheromones	e. g. wasp nest construction

Table 2.1: Varieties and examples of stigmergy

Although a discrimination of quantitative and qualitative stigmergy is relatively easy, most situations in which stigmergy plays a role are likely to involve both types (cf. [CDF+01]). However, the discrimination between these two types will become important for the DIC model presented in Chapter 4.

2.3.2.2 Decrease of Entropy

In the 1970s, the term self-organization itself has been established by later Nobel Prize winner Ilya Prigogine and his colleagues through thermodynamics studies [GP71]. Basically, the idea is that open systems decrease their entropy, i. e. order comes out of disorder, when an external energy is applied on the system. For instance, matter organizes itself under this external pressure to reach a new state where entropy has decreased. Prigogine and his colleagues have identified four necessary requirements for systems exhibiting a self-organizing behavior under external pressure:

- **Mutual Causality:** at least two elements of the system have a circular relationship, each influencing the other.
- **Autocatalysis:** at least one of the elements is causally influenced by another element, resulting in its own increase.
- **Far-from equilibrium condition:** the system imports a large amount of energy from outside the system, uses the energy to help renew its own structures (a kind of autopoiesis), and dissipates rather than accumulates the accruing

disorder (entropy) back into the environment. This fact, which goes back to the second law of thermodynamics[4], allows the system to produce "dissipative structures", which maintain far from thermodynamic equilibrium [NP77].

- **Morphogenetic changes:** at least one of the elements of the system must be open to external random variations from outside the system.

Compared to the concept of stigmergy, there is a fundamental difference here. Whereas in the first case self-organization results from a behavior occurring from inside the system, i. e. from the ants or termites themselves, in the second case, self-organization is the result of a pressure applied from the outside on the system.

2.3.2.3 Autopoesis

In 1979, Varela [Var79] established the notion of autopoiesis (self-production) as the self-maintenance of a system through self-generation of the system's elements, as for instance cells reproduction. Autopoiesis applies to closed systems made of autonomous elements whose interactions self-maintain the system through the generation of system's elements, such as organisms. Varela defined an autopoietic system as being organized as a network of processes of production (transformation and destruction) of elements that

1. through their interactions and transformations continuously regenerate and realize the network of processes (relations) that produced them.

2. constitute it (the system) as a concrete unity in the space in which the elements exist by specifying the topological domain of its realization as such a network.

In the case of autopoesis, self-organization is still different from the two examples above. Autopoiesis applies to closed systems made of autonomous elements whose interactions self-maintain the system through generation of system's elements.

In general, nature provides a vast amount of diverse shapes of self-organization. Further examples can be found in [Fla00, CDF+01, Nun06], for instance. Note, Camazine et al. [CDF+01] also provide four alternatives to self-organization.

2.3.3 Characteristics of Self-Organization

If we narrow down to the disciplines closer related to computer science, we still observe many different definitions. Several authors base their definition of self-organization on the decrease of entropy. For instance, Shalizi and Shalizi [SS03] propose a mathematical model based on Shannon's entropy [Sha48]. In this context, self-organization is characterized as the increase in the amount of information needed for predicting the system dynamics in the future. Similarly, Heylighen and

[4] The second law of thermodynamics states that in an isolated system, entropy can only decrease, not increase.

his colleagues [HJ01, Hey03] argue that the statistical entropy can be used in determining the degree of self-organization of a given system. Parunak and Brückner [PB01] discuss an entropy model for self-organization based on the Kugler-Turvey model [KT86]. Here, information entropy is used to determine the degree of self-organization on the macro-level and the micro-level. Wright et al. [WSDG01] define a self-organized system to have an attractor, i. e. a preferred position for the system, with an intermediate dimension value. For this purpose, they propose a method to measure self-organization, which is based on the property that the entropy can be shown as a function of the attractor's dimension in the state space.

But there also exist other definitions of self-organization that are not based on the notion of entropy but on characteristics of self-organization. For instance, Mühl et al. [MWJ+07] define a self-organizing system a self-managing (i. e. the system adapts to its environment without external control) and structure-adaptive system (i. e. the system establishes and maintains a certain kind of structure, e. g. spatial or temporal, providing the system's primary functionality), employing a decentralized control. Similarly, Di Marzo Serugendo et al. [SGK05] refer to self-organization as a process where a system changes its internal organization to adapt to changes in its goals and the environment without explicit control. Additionally, they distinguish between *strong self-organizing systems*, where there is no explicit internal nor external control, and *weak self-organizing systems*, where the re-organization maybe under internal (central) control or planning (e. g. in the case of a termite queen broadcasting information by pheromone gradients). In the same way Cakar et al. [CMMS+07] distinguish between self-organization with central control and self-organization without central control and discuss the amount of control input allowed from the outside of the system. However, we will see by our approaches in Chapter 4 and Chapter 6 that even these broad definitions are still quite ambiguous and do not cover all classes of self-organizing emergent systems, as they do not distinguish different phases of a running system, which allow the latter e. g. to switch between weak and strong self-organization. Also, the notion of weak self-organization overlaps with the notion of self-adaptation, which indicates that these definitions and classifications are still under way and not settled yet.

The above two categories of definitions, based on decrease of entropy respectively characteristics of self-organization, have different backgrounds. Whereas the purpose of the definitions of the first category is rather on the measurement of self-organization, the purpose of the definitions of the second category is rather on the design and the properties of self-organizing systems. For the purpose of this thesis, the second category is naturally of more interest.

Based on a comprehensive analysis of related definitions in literature, De Wolf and Holvoet [DH04] have identified four characteristics of self-organization that are considered to be most important throughout the literature:

- **Increase in order:** Organization can be described as the arrangement of selected elements so as to promote a specific function. This restricts the behavior of the system in such a way as to confine it to a smaller volume of its state

space (attractor). In essence, organization can be looked at as an increase in the order of the system behavior which enables the system to acquire a spatial, temporal, or functional structure.

- **Autonomy:** A self-organizing system needs to organize without interference, i. e. without an external element imposing it from the outside. Input is still possible as long as the inputs are no control instructions from outside the system. For this purpose it is important to separate the inside from the outside, i. e. to clearly define the boundary of the system.

- **Adaptability or robustness w. r. t. changes:** A self-organizing system is expected to cope with changes in its environment and to maintain its organization autonomously. This requires an adaptive behavior that may take into account past experiences. This also implies the need for the system to be able to exhibit a large variety of behaviors.

- **Dynamical, i.e. far-from-equilibrium:** An essential property of self-organization is that it is an dynamic process. In order to maintain the organized structure influenced by changes, there needs to be a constant dynamic that handles them. In other words, the system needs to be far-from-equilibrium in order to maintain the structure.

A concept tightly linked to self-organization is *emergence*. Briefly, emergence refers to a process by which a system of interacting elements acquires qualitatively new properties that cannot be understood as the simple addition of their individual contributors. This phenomenon is commonly phrased as 'the whole is more than the sum of its parts', but *"it is unlikely that a topic as complicated as emergence will submit meekly to a concise definition"* [Hol98]. A proof of this statement is the history of emergence[5].

2.3.4 History of Emergence

Since the time of ancient Greeks, in more detail the writings by Aristotle in 350 BC [Ari24], conceptual constructs that resemble emergence, such as 'whole before its parts', i. e. to consider an explanation in terms of the global behavior more important than explaining how the system works in terms of local behavior, and 'Gestalt', i. e. a configuration or pattern of elements so unified as a whole that it cannot be described merely as a sum of its parts, can be found in western thought.

In the last two centuries, there were two different schools for studying emergence: the *proto-emergentism* at the end of the 19th century and beginning of the 20th century and the *neo-emergentism* during the 20th century until just now (see [Gol99]). For the proto-emergentists, the process of emergence always remained as a kind of black box. They had few answers, when it came to understanding how emergence

[5]The following historical sketch of emergence is partly based on the work in [Gol99].

itself was at all possible. This means, one could only discern the inputs at the lower-level and the outputs on the higher-level, but not how the input was transformed to the output during emergence. This is exemplified by the notion of emergence by Lewes [Lew75], an English philosopher and proponent of proto-emergentism, in 1875:

> *. . . although each effect is the resultant of its components, we cannot always trace the steps of the process, so as to see in the product the mode of operation of each factor. In the latter case, I propose to call the effect an emergent. It arises out of the combined agencies, but in a form which does not display the agents in action . . . .*

The understanding of emergence (in this quote called "emergent" as noun) is very much like the modern usage, in which nonlinear interactivity leads to novel outcomes that are not sufficiently understood as a sum of their parts. Other proponents of the proto-emergentism school were the animal behaviorist Morgan [Mor26], the philosophers Alexander [Ale90] and Broad [Bro25], the entomologist Wheeler [Whe26], and the mathematician and philosopher Whitehead [Whi79].

As a movement, proto-emergentism died out during the 1930s, and from then until just now a different perspective has been envisaged, called neo-emergentism [Gol99]. This movement tries to open the black box of emergence and to understand as well as to reproduce the process which leads to emergence, by means of high-speed computers, the discovery of pertinent mathematical constructs, and new research methods. As a result, the construct of emergence is acquiring a much surer foundation and usefulness in scientific explanations (cf. [Gol99]).

The neo-emergentism movement has produced various definitions, formalizations, and examples of emergence, which can be found in diverse scientific disciplines today, e. g. philosophy [O'C94, Bed97, BC00, Bed02] and cognitive sciences [Cla98, Ste06], psychology [Gol99], biology [Cam74, BDT99, CDF+01], physics [And72, Cru94b, CM95], cybernetics [Hey03], artificial life [Fai98, RSC99], complexity theory (with its four central schools complex adaptive systems theory [Lan86, Kau96, Hol98], nonlinear dynamical systems theory [New96], the synergetics school [Hak81], and far-from-equilibrium thermodynamics [NP77]), and not least computer science [Bee95, Dys98, PB01, DH04, SGK06, Abb06, MMS06].

2.3.5 Shapes of Emergence

Similar to self-organization, the abundance of appearances of emergence indicates that the understanding and definition of emergence strongly depends on the discipline it appears in. To illustrate this, we will focus on several different appearances of emergence. Thereby we classify these shapes into three categories: emergent shapes that have a desired causal effect on the system, an undesired causal effect, or an insignificant effect, i. e. no causal effect. These examples essentially enhance the fact that there exist multiple shapes of emergence, which are opposed to a general definition.

2.3.5.1 Desired Emergence

Due to evolution, shapes of emergence in nature, in particular in biology, have usually a desired causal effect on the emergent system.

Schools of Fish

Let us take up again the example of schools of fish in more detail. All members of a school move in parallel in the same direction. When a school suddenly changes direction, all its members rapidly respond, moving cohesively, almost in unison, as flawlessly as if they were parts of a single organism. These behaviors suggest that a school possesses special emergent properties on group-level. One such property is the rapid transfer of information throughout the school that enables the entire group to execute swift, evasive maneuvers, at the approach of predators (cf. [Par82]). For instance, flash expansion is an evasive maneuver, in which the school rapidly expands and bursts radically. Another maneuver is the fountain effect, in which a school of small, slow-moving prey outmaneuvers a predator by splitting into two groups, each of which moves in opposite directions and regroups behind the predator (cf. [CDF+01]).

Ant Foraging

Let us also take up again the example of ant foraging in more detail: a moving ant lays some pheromone (in varying quantities) on the ground, thus marking the path by a trail of this substance. While an isolated ant moves essentially at random, an ant encountering a previously laid trail can detect it and decide with high probability to follow it, thus reinforcing the trail with its own pheromone. The collective behavior that emerges is a form of autocatalytic behavior where the more the ants are following a trail, the more attractive that trail becomes for being followed. Thereby a shortest route path from the ant colony to feeding sources and back emerges on the global level (cf. [DG89]).

2.3.5.2 Undesired Emergence

Apart from nature, emergent behavior can have undesired causal effects on a system, which is then termed emergent misbehavior. Even when emergent misbehavior is not inherently bad, it is unpredictable due to the nature of emergence, and unpredictability is bad in many systems – especially in computer systems when it comes about performance. But examples of emergent misbehavior can be found in other fields also (see [Mog06]).

Non-Computer World

When the Millennium Footbridge was opened to significant pedestrian traffic in mid 2000, *"[d]uring the opening day unexpected excessive lateral vibrations occurred,"*

which caused *"a significant number of pedestrians ... to have difficulty in walking"* [DFB+01]. The bridge had to be closed for one and a half years until its engineers analyzed and fixed the problem. The designers had failed to anticipate an effect that could cause the synchronization of individual footfalls, both with each other and with the bridge's natural swaying frequency (cf. [DFB+01]). Another form of emergent misbehavior in the non-computer world are traffic jams.

Hardware World

In huge enterprises or service providers, large numbers of disk drives are mounted on racks. Whereas every single disk drive works fine outside the rack, it turned out that the performance of a single drive can be adversely affected by the vibrations caused by seek activity on neighboring drives in the rack [ADR03].

Network World

The channel or Ethernet capture effect [RY94] is a phenomenon where one user of a shared medium, such as a channel, "captures" the medium for as long as it needs before other users can use the medium. This effect was first recognized in networks using the Ethernet protocol. It occurred in Ethernet links because of the way nodes "backoff" from the link and attempt to re-access it. In the Ethernet protocol, when a communication collision happens, i. e. when two users of the medium try to send at the same time, each user waits for a random period of time before re-accessing the link. However, a user will wait ("backoff") for a random amount of time proportional to the number of times it has successively tried to access the link. The Ethernet capture effect happens when one user continues to "win" the link.

2.3.5.3 Insignificant Emergence

In a few cases, an emergent behavior can also have no causal effect on the system. When we consider stones ordered by the sea, a kind of classification of the stones occur over time. Small, lighter stones are close to the border, while heavy stones are far from it. In this case, the ordering of the stones has no effect at all on the whole system made of the stones and the sea (cf. [Ser06]).

To determine, whether an emergent behavior is desired, undesired, or insignificant, the viewpoint of an observer plays an important role (cf. [GH03]). What under some circumstances can be seen as insignificant, under others can be seen as desired, or even undesired, depending on the purpose and the boundaries of the system, which is ascribed to an observer. Consider the example of schools of fish mentioned above: From the fish's point of view, evasive maneuvers of the school apparently are a desired emergent behavior (even though a fish is not aware of the maneuver of the school). By contrast, from the predators' point of view, evasive maneuvers of schools represent an undesired emergent behavior. From an external viewpoint on the ecosystem of schools of fish and predators, evasive maneuvers are insignificant to the entire system.

2.3.6 Characteristics of Emergence

Similar to self-organization, we can observe a plenitude of approaches to capture the phenomenon emergence in a definition or taxonomy, even if we narrow down to definitions related to (intelligent) computer systems only. Again, the purpose of the definitions and taxonomies depend on the background.

Bedeau [Bed02] distinguishes three kinds of emergence, to understand and explain complex biological and psychological systems, which is essential for using analogies in the computational world. *Nominal emergence* is the simplest and barest notion of emergence, and refers to a macro-level property that is the kind of property that cannot be a micro-level property. It does not explain which properties apply to wholes and not to their parts, but it assumes that those properties can already be identified. For instance, the function of a software system is an nominal emergent property of the underlying code. Strong and weak emergence then add further conditions to nominal emergence. *Strong emergence* adds the requirement that truths concerning the macro-level property are not deducible even in principle from truths on the micro-level. Strong emergence is thus based on the so-called thesis of irreducibility and most closely corresponds to the notion of emergence during proto-emergentism. For instance, Life is a strong emergent property of genes, genetic code, and nucleic/amino acids, whereas Culture in general is a strong emergent property of language and writing systems. *Weak emergence* is between strong and nominal emergence. The central idea behind weak emergence is that emergent causal powers can be derived from micro-level information but only in a certain complex way. In other words, truths concerning the macro-level property are only unexpected given the principles governing the micro-level properties. For instance, foraging behavior of ant colonies as well as flocking behavior of fish and birds are shapes of weak emergence. Weak emergence most closely corresponds to the notion of emergence that is most common in recent scientific discussions of emergence (neo-emergentism), also in computer science. Fromm [Fro05] specifies this basic taxonomy further and splits some of the categories.

Abbott [Abb06] also discusses two kinds of emergence: *static emergence* and *dynamic emergence*. An emergent behavior is called static if its implementation does not depend in time, e. g. hardness as a property of a material (and not a property of isolated atoms). In contrast, an emergent behavior is regarded as dynamic if it is defined *"in terms of how the model changes (or doesn't change) over some time"*. Dynamic emergence can additionally be subdivided into non-stigmatic dynamic emergence (which can be defined by means of continuous equations) and stigmatic dynamic emergence (which involves autonomous entities that may assume discrete states and interact with their environments).

Müller-Schloer and colleagues [MMS06] consider emergence from the analysis viewpoint. They propose a notion of emergence based strictly on measurements, which is called *quantitative emergence*. It is defined as the formation of order from disorder based on self-organizing processes. This definition again builds on Shannon's information theory, in particular on the information-theoretical entropy.

For the purpose of this thesis, again the characteristics of emergent systems are to the fore. Based on a comprehensive analysis of related definitions in literature, De Wolf and Holvoet [DH04] have also identified eight characteristics of emergence that are considered to be most important throughout the literature (see also Figure 2.3 for an illustration):

- **Micro-macro effect:** The micro-macro effect refers to properties, behaviors, structures, or patterns that are situated at a higher macro-level and arise from the local (inter)actions at the lower micro-level of the system. This is the most important characteristic of emergence mentioned in literature.
- **Radical novelty:** The global behavior at the macro-level is novel w. r. t. to the individual behaviors at the micro-level, i. e. the individuals at the micro-level have no explicit representation of the global behavior at the macro-level. As the macro-level behavior is not reducible to the micro-level behaviors of the system, this is termed non-reductionism.
- **Coherence:** Emergent behavior can be specifically identified at the higher level, and consequently a coherence spans and correlates the separate lower level elements into a higher level unity, i. e. correlations between elements are needed to reach a coherent whole, which is also called 'organizational closure'.
- **Interacting elements:** Without interactions, interesting macro-level behaviors will never arise. Simple parallelism is not enough to yield emergent behavior.
- **Dynamical:** Emergent behavior arises as the system evolves in time. The appearance of emergent behavior can be related to the appearance of new attractors in dynamical systems.
- **Decentralized control:** Whereas the actions of single elements are controllable, the whole system is not directly controllable. In particular, there is no central control, i. e. no single element of the system directs the macro-level behavior. This is a direct consequence of the radical novelty that is required for emergence.
- **Two-way link:** From the micro-level to the macro-level, the elements give rise to an emergent structure. In turn, the emergent structure influences the elements, i. e. higher level properties may have causal effects on the lower level.
- **Robustness and flexibility:** The characteristic of decentralized control and the fact that no single element can have a representation of the global emergent behavior implies that such a single element cannot be a single point of failure. Increasing damage will decrease performance, but with a 'graceful degradation', i. e. the quality of the output will decrease gradually, without sudden loss of function.

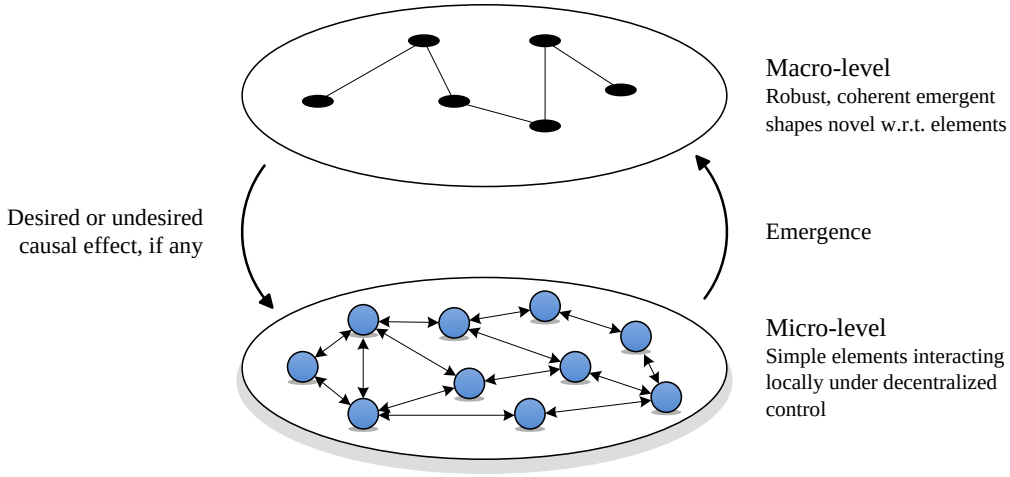


Figure 2.3: Micro-level, marco-level, and emergence

2.3.7 Self-Organizing Emergent Systems

To summarize, the essence of self-organization is an adaptable behavior that autonomously acquires and maintains a structure, i. e. an increased order, statistical complexity, etc. The essence of emergence is the existence of a global behavior that is novel w. r. t. the constituent elements of the system. The main similarity between the concepts is that self-organization and emergence are both dynamic processes arising over time. The individual elements are all "active" as well as locally interacting and may have their own objectives and carry out their respective tasks. Both are robust, in more detail, whereas self-organization is robust w. r. t. environmental changes and is able to maintain the increased order, emergence is robust w. r. t. the flexibility of the elements that cause the emergent behavior, i. e. the failure of one single element will not result in a complete failure of the emergent behavior.

De Wolf and Holvoet [DH04] emphasize self-organization and emergence as different characteristics of a system. According to the authors, both concepts can exist in isolation or together. A self-organizing system without emergence controls itself without external interference, but lacks central properties of an emergent system such as radical novelty, micro-macro effect, flexibility w. r. t. the elements, and decentralized control. For example, a system where there is a single controlling element that directs the global behavior (i. e. there is no decentralized control) needs an explicit plan in that controlling element. A (weak) self-organizing process can re-elect a controlling element when other elements become more appropriate for the job, but there is no radical novelty.

On the other hand, according to the authors, there are also systems exhibiting emergence without self-organization. An example given is a gas material that has a certain volume in space, which is an emergent property that results from the interactions, i. e. attraction and repulsion, between the individual particles. The gas

is in a stationary state. The statistical complexity remains the same over time, i. e. the particles can change their places but the amount of structure remains the same over time. This is a system whose initial conditions are enough to exhibit emergent properties but no self-organization is attributed to such a system, since the system is stationary. Stones ordered by the sea are another example, as mentioned above.

While a separate discussion of the terms self-organization and emergence is definitely valuable, it is questionable whether emergence without self-organization is actually relevant for computer systems. In most systems that are considered in literature, emergence and self-organization occur together. Thus, from the viewpoint of computer science, in particular when considering self-organizing systems for self-management, only natural or artificial systems of the latter category are of interest, where we usually have decentralized control realized under self-organization leading to emergent behavior. We take up this position in this thesis, too.

Future complex computer system often require to keep the individual elements relatively simple, e. g. for scalability reasons. Simple individuals cannot direct such a complex system, so the global coherent behavior should emerge from the self-organizing interactions between the individuals. Similarly, because of the complexity it is sometimes impossible to impose an initial structure on such a system that results in an emergent property. The only possibility to get a coherent behavior at the macro-level is to let that behavior arise and organize autonomously, which implies self-organization. Thus, the combination of self-organization and emergence is a promising approach to engineer a coherent, self-managing behavior for complex computer systems. This leads us to the definition of self-organizing emergent systems in the context of self-management, as we will understand and use this term throughout the rest of this thesis. The definition is partly based on the definitions of self-organization and emergence found in [DH04] and [SGK05].

Definition 2.1 (Self-organizing emergent system)
A self-organizing emergent system dynamically acquires or maintains a structure without external control. Local (inter)actions of system elements at the micro-level result in emergent shapes at the macro-level that are novel w. r. t. the elements and either desirable or undesirable w. r. t. the system.

This definition is consistent with the properties commonly considered to be relevant for self-organization and emergence (see [DH04]). In this definition, self-organization refers to a *dynamic process*, which is *far-from-equilibrium*. The intended 'structure' can be of spatial, temporal, or functional nature and is acquired or maintained w. r. t. dynamic changes in the system's environment, which presumes an *adaptivity and robustness* of the process. 'Without external control' refers to the *autonomy* of the system and the absence of direction, manipulation, interference, or involvement from outside the system.

The definition expressly underlines that 'emergent shapes' at the macro-level, which can be emergent properties, behaviors, structures, patterns, or functions, are a result of self-organization, i. e. self-organization at the micro-level is the cause, as e. g. considered in [CDF+01, Hey03, MMTZ06], but not an effect of self-organization,

i. e. self-organization is an emergent property, as e. g. considered in [PB01]. Because the emergent shapes have to be novel w. r. t. the interacting elements, weakly self-organizing systems having an internal central control cannot be referred to as self-organizing emergent systems (even though they are still self-organizing). In order to decide, whether the increase in order is produced on its own, i. e. 'self', system boundaries play an important role. Consequently, a self-adaptive system can be considered as weakly self-organizing, if the subsystem executing the control loop is within the defined system boundaries.

The definition also classifies the effects of emergent shapes w. r. t. the system. As explained in Subsection 2.3.5, emergent shapes can be desired, e. g. evasive maneuvers in schools of fish, or undesired, e. g. unexpected vibrations on the Millennium Footbridge (insignificant effects, e. g. stones ordered by the sea, are not of interest and thus not included, as mentioned). We think that the importance of these effects to the design and operation of self-organizing emergent systems with regard to self-management justifies their inclusion into the definition.

2.4 Multi-Agent Systems

In general, there exist different technologies to model and realize self-organizing emergent systems as defined in the previous section, e. g. by cellular automata [Gut91], neural networks [Hay98], or multi-agent systems [Woo09]. However, multi-agent systems have been identified as the enabling technology for a plethora of application domains (cf. e. g. [PB01, MRF+03, SKRZ04, BSKN05, SGK05, SGK06, BSHZ06, BHJY07]), which is why we will base our approaches on this technology, too. A multi-agent system (MAS) is a particular class of computer systems composed of multiple interacting computing elements, called agents. The latter usually have the following characteristics (cf. [JSW98]):

- **Autonomy:** Agents in a MAS are considered as autonomous or semi-autonomous hardware or software systems that operate in a asynchronous manner without the direct intervention of humans or others.
- **Local views:** Agents only have local, subjective views of the entire system. Usually, the systems are too complex for an agent to acquire or maintain a global view, so it can only have incomplete information of the system.
- **Decentralized control:** As a direct consequence of the second characteristic, in a MAS usually no agent has global control over the entire system.
- **Decentralized data:** Data is fully decentralized and distributed over the agents of the system and the environment.

Due to these characteristics, MASs are generally considered as an eligible technology for accurately modeling and implementing self-organizing emergent systems. The field of MASs has been studied since about 1980, and has only gained widespread

recognition since about the mid-1990s. Nonetheless, the debate on what constitutes an autonomous agent and how to define it is still under way. Wooldridge and Jennings [WJ95] attribute an autonomous agent with three essential properties:

- **Social ability:** Agents interact with other agents (and possibly humans) via some kind of agent-communication language (ACL)
- **Reactivity:** Agents perceive their environment and respond in a timely fashion to changes that occur in it.
- **Pro-activeness:** agents do not simply act in response to their environment, they are also able to exhibit goal-directed behavior by taking the initiative.

Although these properties are cited very often in literature, they do not represent a general characterization of an agent, because they require agents to interact using an ACL, e. g. KQML [FMM94] or FIPA-ACL [FIP02b]. But there exist several types of autonomous agents that only interact through the environment, not in need of ACLs. A more general definition of an agent is given by Wooldridge [Woo09]: *"An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives."* An agent is in possession of a repertoire of autonomous actions that it can execute to modify the environment. Executed actions effect changes in this environment, which can be sensed by sensors of an agent, either physical sensors (in the case of physically embodied agents situated in a part of the real world) or software sensors (in the case of computational agents such as simulated or software agents).

The field of MAS research did not emerge from a vacuum but has commonalities and differences with other research fields. For a long time it was common to refer to MAs as a subfield of Artificial Intelligence, namely Distributed Artificial Intelligence, see e. g. [Wei99]. But it has become a common practice to define the endeavor of AI itself as one of constructing an intelligent agent, see e. g. [RN02]. This is, AI is more concerned with the *components of intelligence*, i. e. the ability to learn, to plan, to understand concepts, etc. In addition, classical AI has largely ignored the social aspects of agencies [Woo09]. Another related research area is for instance game theory. Tools and techniques of game theory have found many applications in computational MASs, see e. g. [SLB09], although many of the solution concepts developed in game theory tend to be descriptive concepts without a view to computation [Woo09]. Franklin and Graesser [FG97] discuss the differences between agents and arbitrary computer programs, whereas Wooldridge [Woo09] in addition discusses differences between MASs and distributed/concurrent systems, economics, and social science.

2.4.1 Generic Agent Architectures

Based on the agent theory [WJ95], a spectrum of generic agent architectures have appeared (see for instance [Oss99, Sch01]):

- **Deliberative agents:** A deliberative agent, on the one side of the spectrum, maintains an internal representation of the world in which it lives, i. e. it has an explicit mental state. Thus, this type of agent is also referred to as *mental* or *rational agent*. The representation is an explicit symbolic model, which can be modified by some kind of symbolic reasoning, i. e. a planner reasons on this model and decides which actions to realize, while the agent uses sensor data in order to update this model. One of the best known deliberative agent architectures are BDI agents, which are based on the BDI (belief, desire and intention) theory [RG95]. Beliefs represent the informational state of the agent, i. e. its knowledge about the world, desires (or goals) represent the motivational state of the agent, i. e. objectives or situations that the agent would like to accomplish or bring about, and intentions represent the deliberative state of the agent, i. e. what the agent has chosen to do. Even though they are not a conceptual ingredient of the BDI theory, plans are essential, representing sequences of actions that an agent can perform to achieve one or more of its intentions.

- **Reactive agents:** A reactive agent, on the other side of the spectrum, works in a hard-wired stimulus-response manner. Instead of any symbolic representation of the world and any abstract reasoning, a reactive agent makes its decisions directly based on the input of its sensors. Hence, the decisions are usually based on a very limited amount of information along with simple situation-action (if-then) rules. The focus of this agent type is rather on achieving robustness instead of optimality. One of the best known reactive agent architectures is the so-called subsumption architecture [Bro86].

- **Hybrid agents:** In between the spectrum, a hybrid agent represents a unification of deliberative and reactive agents in order to surmount their respective weaknesses. Whereas reactive agents very seldom implement goal-directed behavior, deliberative agents are mostly based on general-purpose reasoning mechanisms, which are not tractable and much less reactive. Essentially, hybrid agents are steered by their simple routines reacting to basic stimuli. However, a deliberative module controls the reactive one when it wants to perform stimulus-free actions (like reasoning) or to change long-term goals. An important example is the hybrid architecture InterRAP [Mül96].

Apparently, these three agent architectures differ from one another in their view of the intra-agent aspects. However, these agent architectures are idealized ones. In practice, there exist much more architectures mixing different aspects of these architectures. For example, the agent architecture used for our coordination approach in Chapter 4 resides between purely reactive agents and hybrid agents, and comes close to the notion of situated agents [WH04]. In situated MASs the agents work together locally to solve complex global problems. They are characterized by the presence of an explicit spatial structure in which agents act, i. e. every situated agent has an explicit position in the (distributed) environment. Situated agents live and

act in the presence and their decision making is not based on extensive reasoning upon mental states. They do not use long-term planning to decide what action sequence to execute next, but rather use computationally efficient behavior-based action selection mechanism to select situated actions they will perform. Behavior-based action selection is driven by stimuli perceived in the environment as well as stimuli internal to the agent. The situated actions hence are selected on the basis of an agent's position in the environment, the state of the world it perceives, and only a limited amount of its internal state. The internal state will only be employed for decision making if it relates to (1) general static information of the system, e. g. fixed priority rules, (2) dynamic information related to the agent's current context, e. g. a temporal agreement for collaboration with an agent on the same position, or (3) issues internal to the agent, e. g. a threshold value used as a switch for changing roles (cf. [Wey06]). Due to such an action selection mechanism a situated agent is able to respond rapidly to dynamic and changing circumstances. In contrast to deliberative agents, situated agents also do not emphasize an internal modeling of the environment. Instead, they favor to employ the environment itself as a source of information.

Also, a MAS has not to be based on a single agent architecture exclusively. As we will see in Chapter 6, we use different types of agent architectures to realize a self-managing systems combining self-organization with self-adaptation.

2.4.2 Formal Definitions

Due to the large range of agent types and architectures, various descriptions of an agent have been proposed, without, however, reaching a commonly accepted definition. A very general but concise definition of an agent that can be instantiated to most of the views of agents in literature is provided by Denzinger and colleagues (cf. [KD06]):

Definition 2.2 (Agent)
An agent Ag is defined as a quadruple

$$Ag = (Sit, Act, Dat, f_{Ag})$$

where

- Sit is a set of situations Ag can be in
- Act is the set of actions Ag can perform
- Dat is the set of possible values that Ag's internal data areas can have
- $f_{Ag} : Sit \times Dat \rightarrow Act$ is the decision function Ag uses in order to determine its next action

Consequently, they define a MAS (see Figure 2.4) on a very high level as follows:

Definition 2.3 (Multi-agent system)
A multi-agent system MAS is defined as a pair

$$MAS = (A, Env)$$

where

- A is a set of agents $Ag_1, \ldots Ag_m$
- Env is a common environment (or at least parts of it) the agent in A share in order to interact with each other

The agents in A might all have different sets of situations, actions, and internal data area values and they naturally can also have different decision functions. The actions of the agents in A might change the environment (or it can change on its own) and therefore Env consist of a set of environment states.

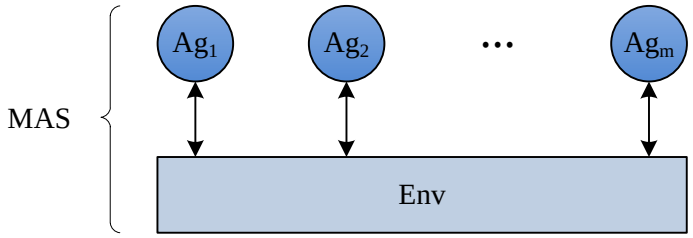


Figure 2.4: Components of a multi-agent system

Based on these two definitions, the first three components of an agent can be more structured by distinguishing between parts dealing with the agent in the environment itself and parts dealing with other agents. More formally, an element sit of Sit has the form $sit = sit_{Env}sit_{Ag}$, where sit_{Env} describes the environment the agent acts in without the other agents and sit_{Ag} provides the information about other agents. Note, the set Sit of an agent can be just a view on the environment, i. e. an element of Sit can contain less information than an element of Env. Act can be divided into the sets Act_{own} of the agent's actions not dealing with other agents and Act_{co} of its communication and cooperation actions. Dat can be further structured into the set of data areas that contain information about the agent itself (Dat_{own}), areas that contain sure knowledge about other agents (Dat_{sk}), and areas that contain assumed (and therefore unsure) knowledge about other agents (Dat_{ak}).

For an external observer lacking initial knowledge about an agent, the agent's decision function appears to be a function $g_{Ag} : Sit \rightarrow Act$, i. e. an observing agent can only perceive actions and situations, but not the internal decision mechanisms that drive an agent's decision making. The extent of the difference between an actual f_{Ag} and apparent g_{Ag} distinguishes between reactive and deliberative (or proactive respectively knowledge-based) agents. An agent Ag is reactive if its Dat has negligible influence on its decision making, and deliberative if its Dat has significant

influence on its decision making. Consequently, for situated agents *Dat* only has limited influence. Finally, the realization of *Ag*'s actual f_{Ag} depends on the agent's architecture.

In contrast to e. g. [Syc98], these very general definitions make no assumptions about the way a MAS is controlled, the communication and interaction patterns, or the information available to each agent. Due to this generality, these definitions also apply to several classes of systems, not only artificial ones but also human or animal systems. However, due to the abundance of proposed agent paradigms and models, any stricter definition will automatically exclude some of them. Thus, we will base our concepts presented in Chapter 4 and Chapter 6 on these definitions.

2.5 Related Research Areas

As mentioned in the previous sections, the tackled problems and challenges of this thesis emanate from the areas of self-adaptive and self-organizing systems, respectively. Although these classes of systems constitute separate research areas, which may converge in the area of self-managing systems, they are related to other research areas as well. In virtue of the tackled problems and challenges this thesis is mainly related to two further research areas, which are Autonomic Computing (Subsection 2.5.1) on the industrial side and Organic Computing (Subsection 2.5.2) on the academic side.

2.5.1 Autonomic Computing

Autonomic Computing (AC) is a term as well as a paradigm coined and promoted in the first instance by IBM since 2001 [Hor01]. The overarching goal of AC is to realize computer systems and applications that can manage themselves in accordance with high-level guidance from humans [PH05]. The need and justification for AC is based on the permanently increasing complexity of today's computer systems accompanied with the escalating costs for managing these systems and thus based on the enduring validity of Moore's law [Moo65]. Because the AC Initiative is driven by an IT vendor, its focus is on enterprises and their respective needs in the first instance, e. g. reducing TCO of data centers, efficient power management, etc.

More specifically, AC aims at designing and building systems that are self-managing, i. e. self-configuring, self-optimizing, self-healing, and self-protecting, forming an autonomous and ubiquitous computing environment that completely hides its complexity and provides the user with an interface that exactly meets her/his needs [SPTU05]. Although initially equipped with these four self-* properties[6] and additional major characteristics such as knowing itself and being context-aware, open, and adaptive [Mur04], numerous additional self-* properties became attributed to AC systems in the following years. A non-exhaustive list includes self-governing, self-

[6] Sometimes also referred to as self-CHOP properties (self-*c*onfiguring, self-*h*ealing, self-*o*ptimizing, and self-*p*rotecting)

adapting, self-organizing, self-recovering, and self-diagnosing of faults (cf. [SPTU05]), self-planning, self-learning, self-scheduling, self-evolving, self-regulating, self-correcting, self-administering, self-monitoring, self-adjusting, self-tuning, self-aware, self-modeling, self-assessing of risks (cf. [Tia03]), and self-assembling [TCW+04], for instance. Summarizing these self-* properties, an AC system is expected to manage itself on its own without conscious recognition or significant effort on the side of the users, allowing the latter to concentrate on what they want to do without worrying about how it has to be done.

This management paradigm is inspired by the human autonomic nervous system (ANS), which is the part of the peripheral nervous system that acts as a control system. As a silent guardian the ANS constantly regulates and controls the internal organs and vegetative functions of the body such as breathing, heart beat adjusting, body temperature control, digestion, fending off viruses, circulation of the blood, or hormone production. The ANS monitors changes inside and outside the body, integrates sensory inputs, and effects appropriate response without any conscious recognition or effort by the human itself [Enc08]. In the same way the ANS manages the human body, AC systems are intended to have the ability to manage, repair, and protect themselves. Apparently, this approach corresponds to exogenous self-management (see Section 2.2).

To support this vision of AC [KC03], IBM provides an architectural blueprint for AC [IBM06]. Although there exist other architectural approaches for AC as well, see e. g. [WHW+04], the blueprint by IBM is the best-known approach. It organizes an AC system into several building blocks that are the architectural representations of the components in an AC system working together to provide autonomic capabilities. The building blocks can be composed in different ways. A possible topology for composing these building blocks in a hierarchical way is represented by the generic AC reference architecture (see Figure 2.5).

The lowest layer contains the different system components (servers, databases, applications, etc.), called *managed resources*, that make up the IT infrastructure of an enterprise or organization. Managed resources may already have embedded self-* properties themselves. The next layer incorporates consistent, standard manageability interfaces for accessing and controlling the managed resources, implemented by so-called *manageability endpoints* or sometimes *touchpoints*. A particular resource may have one or more *resource autonomic managers* or *touchpoint autonomic managers*, each implementing a control loop. Figure 2.5 illustrates this by depicting a touchpoint autonomic manager for every broad self-* property in the third layer. The fourth layer contains *orchestrating autonomic managers* that orchestrate the touchpoint autonomic managers to deliver system-wide autonomic capabilities by incorporating control loops that have the broadest view of the overall IT infrastructure. The top layer illustrates a *manual manager*, which is the architectural representation of the human activity typically involving a human administrator using a management console, collaborating with or orchestrating other autonomic or manual managers. The various manual and autonomic manager layers can obtain and share knowledge via *knowledge sources*. A knowledge source is an implemen-

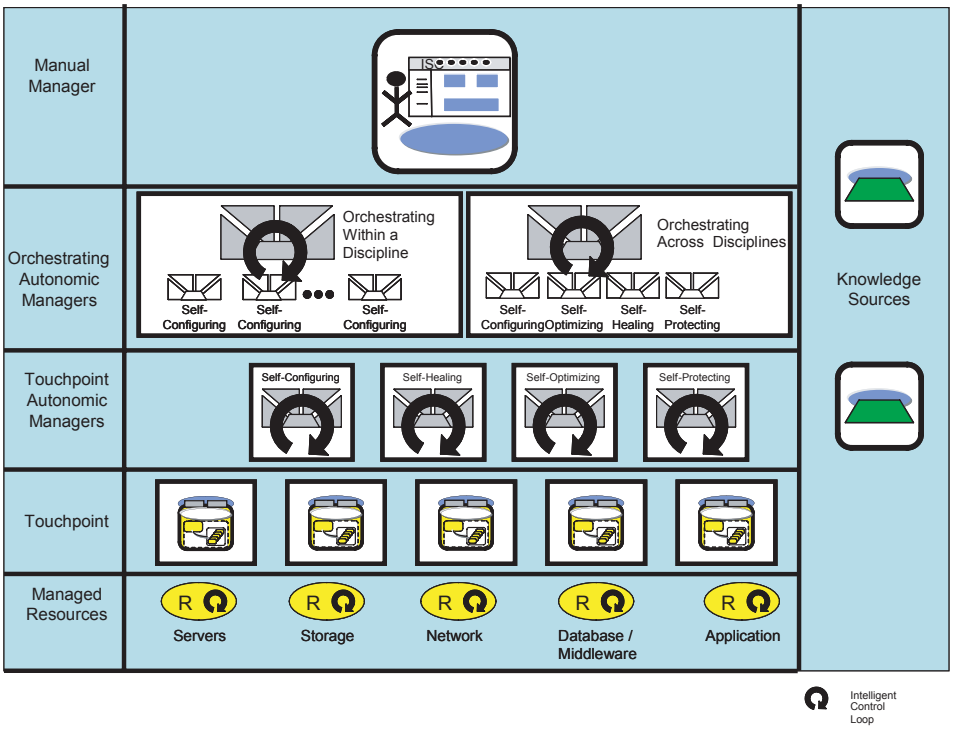


Figure 2.5: Autonomic Computing reference architecture [IBM06]

tation of a registry, dictionary, database or other repository that provides access to knowledge such as management data with defined syntax and semantics. All of these AC building blocks may be connected e. g. by an enterprise service bus that directs the interactions between them.

Apparently, central to the provisioning of system-wide self-managing capabilities in AC systems is the implementation of a feedback control loop by an autonomic manager (AM). We will investigate this implementation in an AM later in Section 5.3 more specifically.

2.5.1.1 Levels of Autonomic Maturity

There is, however, a general consensus in industry that the creation of fully self-managing systems will not happen overnight and will require considerable investment and cross-industry cooperation. Incorporating self-managing capabilities into an IT infrastructure is more an evolution than a revolution. The alternative today is between an automation of some management functions and fully manual management [BF03]. In the first architectural blueprint for AC [IBM03], IBM has defined five autonomic maturity levels:

- **Basic:** point solutions provide management input to humans for their analysis, action planning, and initiation.

- **Managed:** information is consolidated and filtered. Humans still determine, plan, and execute management actions.
- **Predictive:** situations are detected automatically and recommended actions are presented to humans for approval.
- **Adaptive:** performance parameters are adjusted automatically without human intervention. Security and failure actions may also be taken automatically.
- **Autonomic:** humans provide guidance to automatic systems in terms of business objectives.

In the fourth architectural blueprint for AC, these levels have been refined to the AC adoption model as depicted in Figure 2.6, which describes the evolution towards more highly autonomic capabilities of an IT infrastructure. This model focuses on businesses that want to calibrate and increase the degree of autonomic capability that their current infrastructure and organization has. Thus, the model is spanned by three dimensions: the "functionality" dimension, the "control scope" dimension, and the "service flow" dimension. Whereas the functionality dimension characterizes the extent of automation of the IT and business processes by five levels (which are comparable to the five maturity levels mentioned above), the control scope dimension characterizes what is being managed. Finally, the service flow dimension captures the combination of IT management process activities that are being performed.

2.5.1.2 Application Areas

In recent years, the AC paradigm and its corresponding technologies was particularly used for research in the following key applications areas (cf. [HM08]):

- **Data Centers, Clusters, and Grid Computing Systems:** Installing and maintaining these kinds of distributed systems (which can potentially span to worldwide) is very complex, because they are essentially high performance, heterogeneous, distributed clusters of computers used to run anything from scientific to business applications for various users. Hence it is an excellent application area for AC solutions. The two key areas of autonomic research in here are dynamic resource management, which has received the most attention, and systems administration. Typically data centers, service centers, or grids are described as a service-oriented architecture (SOA) and therefore quality of service (QoS), which could be formalized in service level agreements (SLA), determines the resource selection options (see [MB07]). This is further complicated by the fact that the dynamic nature of the system's load, up-times, uncertainties, etc. have to be taken into account not only when initially allocating resources, but while the application is using those resources. Typically this has lead to a grading of service levels as Platinum, Gold, Silver, and Bronze. However, what these levels actually mean depends on the system in question and their users, and varies greatly throughout the literature

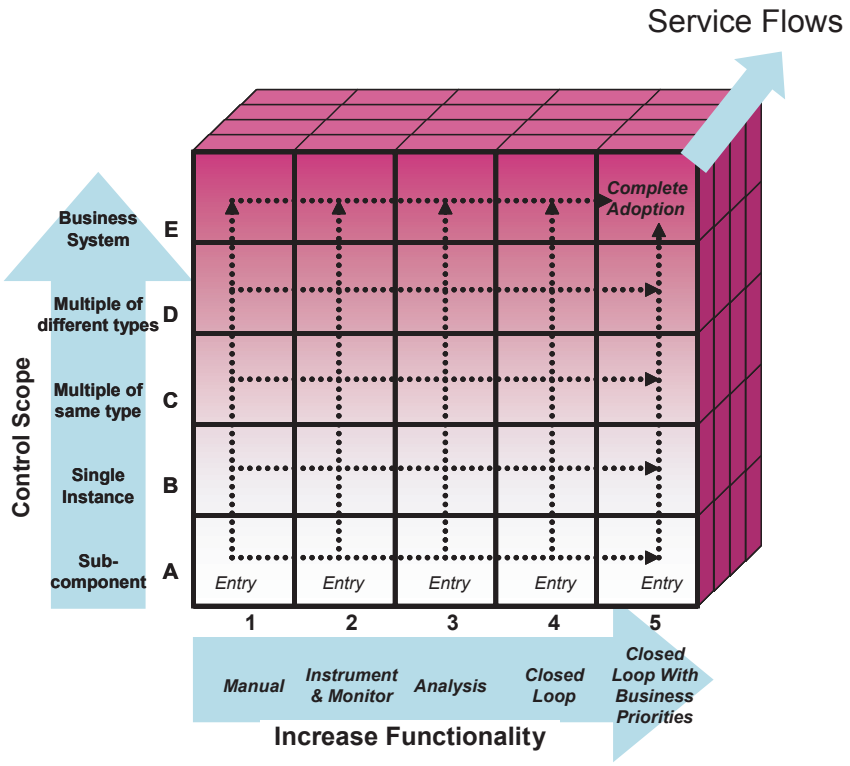


Figure 2.6: Autonomic Computing Adoption Model [IBM06]

(see e. g. [ACMS06]). For systems administration, in many examples (see e. g. [BML+05, ZYK07]) a profile of the human operator behavior and its relation to the system's tuning constants or operator action, is obtained. Effectively, from these profiles a set of actions and responses are obtained. Consequently, a set of policies can be derived to drive the autonomicity. Thus, when a policy rule is broken, an action is taken that is determined by what was learned by the profile.

- **Power Management:** It is estimated that power equipment, cooling equipment, and electricity together are responsible for 63% of the TCO of the physical IT infrastructure of a data center [Ame03]. Such statistics served as a motivation for self-adaptive systems that optimize the resource management in data centers not only in terms of performance metrics but also in terms of the power that a given algorithm or service will consume on a given infrastructure. Whereas earlier work focused on the processor's power consumption for individual server nodes [KAH04] or power allocation of server clusters [FF05], more recent work focuses on models that take memory, network, and IO consumption [KH07, KHY08] as well as heat management [MCR06] into account. It was demonstrated that such autonomic power and performance management

in e-business data centers is able to achieve 72% savings in power consumption while maintaining performance as compared to static power management techniques [KHY08]. Autonomic solutions based on limited lookahead control were able to save on average 22% in power-consumption costs of a server cluster over a twenty-four hour period when compared to a system operating without dynamic control while still maintaining QoS goals [KKH+09]. Autonomic solutions based on MASs can achieve power savings of more than 25% in blade centers [DKL+08].

- **Ubiquitous Computing:** This area concerns the building of intelligent environments from a number of, potentially, heterogeneous devices such as sensor nodes, PDAs, PCs, etc. having small amounts of resources and power [Wei91]. The sheer complexity of installing and maintaining ubiquitous computing systems and keeping them running in a robust way, easily lends itself to AC. The application areas for these systems range from monitoring vineyards [BBB04] to looking after the elderly in the home [HMH07]. The research concentrates in particular on autonomic wireless sensor networks (WSN). This is motivated by the key assumption that the main cost of WSNs lies in driving the wireless equipment and not the sensors or CPUs. Thus, routing data from one node to another has received much attention, where the aim is to minimize the time to deliver a packet whilst minimizing the energy consumed in delivering it. Depending on the resources and power of devices, either "middleware" solutions running a kind of AM have been investigated for higher end devices (e. g. [HM05, SBH+06, TSTN06]), whereas more lightweight, "emergent" solutions have been investigated for lower end devices (e. g. [PMN05, HMH07]). In these latter approaches, autonomicity is embedded into the core of the sensor nodes thus providing for self-management from the lowest level.

2.5.1.3 Research Challenges

Before AC systems become a solution to the complexity problem, certain research challenges have to be tackled (see also [Mur04, HMG05]). Kephart [Kep05] divides this research space into three basic parts:

- **Autonomic elements:** This part focuses on research directed towards improving the self-managing capability of specific components, research on technologies that are generally applicable to autonomic elements, and research on the internal structure of autonomic elements, as well as tools to create them.

- **Autonomic systems:** This part focuses on research on generic technologies that entail interactions among multiple autonomic elements to achieve system-level goals, research on system-level architectures that effectively govern interactions among autonomic elements, and research on fundamental science of large-scale autonomic computing systems, addressing questions of learning, stability, control, and emergent behavior in multi-agent systems.

- **Human-computer-interactions:** This part focuses on research on present and future interactions between human administrators and other users and self-managing systems as well as research on methods for eliciting high-level policies from people and representing and appropriately transforming those policies within autonomic systems.

Apparently, the problems and challenges tackled by this thesis intersect with the second part of the AC research space, namely research an autonomic systems. There is an initial interest in the AC community to make AC systems more and more decentralized and less deterministic, as well as to engineer, exploit, and control emergent features (see e. g. [ERA+03, BJM04, ABI07]). This is why self-organizing emergent systems in this community are sometimes also referred to as decentralized AC systems.

2.5.1.4 Further Industrial Initiatives

Although IBM has certainly started the most well-known industrial initiative tackling the management of IT complexity, other IT vendors have started similar programs with comparable objectives. However, in contrast to AC, none of them has coined this research area such as IBM's initiative.

Since 2003, Microsoft has been proposing its *Dynamic Systems Initiative* (DSI) [Mic03]. For Microsoft, dynamic systems are systems designed to enable businesses and the people in them to meet dynamic demands with a quick and effective response [Mic07]. The DSI is Microsoft's ten year strategy for developing and delivering dynamic systems technologies that enable businesses and the people in them to be more productive and to better adapt to dynamic business demands. The three architectural elements of this strategy are *design for operations* (to capture the diverse knowledge of people), *knowledge-driven management* (that enables systems to capture desired states of configuration and health in models based on business priorities and demands), and *virtualized infrastructure* (to achieve greater agility and leverage existing infrastructure by consolidating system resources into a virtual service pool) (cf. [Mic07]). Remaining challenges in this program are e. g. building up a library of knowledge in management models and to deliver the benefits of dynamic systems across heterogeneous environments.

Initially started in 2003 under the name of *Adaptive Enterprise*, Hewlett-Packard (HP) has renamed its program into *Adaptive Infrastructure* [Hew08], focusing on next-generation data centers. HP's vision of a next-generation data center (NGDC) is a 24/7 environment that is highly automated and establishes the data center as a supply chain for IT services. The key methodologies/enablers of an adaptive infrastructure are *standardization* (systems – including servers and storage – will be standardized and simplified around pre-set configurations mapped to the main types of service usage), *management* (through a "single-view" IT services control room that requires few people to manage the delivery of IT services following a standard process), *virtualization* (separates the hardware owner from the applica-

tion owner allowing operations, configurations, monitoring operations, and tool sets to be homogenized), *automation* (dynamical reallocation of resources to meet changing business needs), *energy efficiency* (focusing on chip efficiency and data center cooling in a holistic manner), and a *repeatable shared services pattern* (a small number of shared IT resource configurations, whereby infrastructure, information, and applications are delivered as services). HP has even defined an adaptive infrastructure maturity model comprising five stages of maturity (see [Hew07]).

In addition to these two programs, in the last few years there have been announced and sometimes partly abandoned some more programs or visions, more or less successfully. Sun has started its N1 program [Sun02b] in 2002, which is part of Sun's management software focusing on enabling IT managers to respond quickly to changes in business requirements and customer needs, focusing on NGDCs as well. Intel had the vision of Proactive Computing [Ten00, WPT03] in 2000, aiming to built computers that will anticipate our needs and sometimes take action on our behalf.

In 2002, Forrester Research proposed the term Organic IT [Gil02] to describe all the industrial efforts executed in this area. They defined Organic IT as *"computing infrastructure built on cheap, redundant components that automatically shares and manages enterprise computing resources – software, processors, storage, and networks – across all applications within a data center."*

2.5.2 Organic Computing

Closely related to the objectives of AC is the Organic Computing (OC) initiative [OCI05], which is by contrast an academia-driven research area and thus is not covered by the umbrella of Organic IT, as one could assume. The term 'Organic Computing' emerged from a workshop held in 2003 on 'Hot Topics in Computer Engineering' by a GI/ITG special interest group on computer engineering and architectures of computing systems. The results of the workshop have been summarized in a position paper [VDE03], which built the basis for a priority program on Organic Computing [Ger05] that started in 2005.

In general, OC is based on the intuition that in near future we will be surrounded by large collections of autonomous elements, which are equipped with sensors and actuators, aware of their environment, able to communicating freely, and capable of organizing themselves in order to perform and fulfill the actions respectively services that are required. In contrast to the business perspective of AC, OC claims to have a strong orientation towards human needs, aiming to construct systems as robust, safe, flexible, and trustworthy as possible. As these systems therefore are expected to exhibit life-like properties, such as acting more independently, flexibly, and autonomously, these systems are called 'organic'. Consequently, an OC system is a technical system, which adapts dynamically to the current conditions of its environment while satisfying humans needs. Example application areas are the control of homes, e. g. with respect to energy consumption, assistance in driving and maintaining cars, monitoring of health and alerts in case of dangerous conditions,

supervising children on their way to school, or organizing shopping lists (cf. [Sch05]). Similar to AC systems, OC systems hence have emerged as a challenging vision for future information processing systems, and are thus likewise attributed with properties such as being self-organizing, self-configuring, self-optimizing, self-healing, self-protecting, self-explaining, and context-aware.

2.5.2.1 Research Challenges

The OC community has identified several research challenges for OC systems, too (see e. g. [MSvdMW04, Sch05]). The challenges became part of the grand research challenges of computer engineering [BBF+08] with regard to OC techniques. However, the boundaries of these challenges are not sharp and hence some of these challenges overlap with each other, while some of them additionally overlap with the research challenges for AC:

1. **Control of self-organization and emergence:** Because the effects of self-organization and emergence can be positive as well as negative, these effects have to be detected and controlled during the operation of OC systems. Mechanisms are required that prevent negative emergence (emergent misbehavior) as well as at the same time produce positive emergence.

2. **Organic Computing architectures:** Standardized building blocks are required that monitor, analyze, and coordinate dynamic and adaptive system elements, evaluate alternative configuration, cooperation, and conflict handling processes, as well as execute reconfigurations. On the long run, these control bricks might be more complicated compared to he underlying productive system.

3. **Self-* properties:** The requirements on self-configuration, self-optimization, self-healing, and self-protection have to be implemented by a holistic architecture to achieve the self-management of OC systems. This architecture has to be completed by methods for self-monitoring and self-controlling.

4. **Safety, robustness, and trustworthiness:** Adherence to certain behavior guarantees in safety-critical application areas, integrity of data in personalized services, or reliability of behavior assertions are essential for OC systems, despite their unforeseeable behavior due to the micro-macro gap.

5. **Engineering methodologies, tools, and mechanisms:** To engineer OC systems in a reasonable way, new methodologies, tools, and mechanisms are required that allow for a self-organizing, emergent, and adaptive behavior of the system elements within certain boundaries or constraints.

6. **Understandability for humans:** Mechanisms and techniques for self-explanation are required that put across the complex and dynamic organization of a OC system for humans respectively provide abstractions to cover the aspects of usability and trustworthiness.

7. **Learning mechanisms:** System optimization and failure predication require learning mechanisms that operate at runtime. However, the learning space has to be restricted to prevent unwanted or harmful system states.

8. **Inspirations from brain research and bionics – activity, motivation, and emotions:** Some elements of a OC system have to be active and have to pursue own goals, in order to achieve the self-CHOP properties. According to the OC community, their behavior can be described with properties such as motivation, desires, or self-confidence. Thereby, analogies from the human brain or from bionics are expected to help.

9. **Reduction of complexity:** Using the reduction of complexity as a design criteria may be an alternative approach. Thereby, a system is reduced to the bare necessities to fulfill the required tasks with minimal interaction, in order to keep the system simple and manageable. A combination of simple subsystems to more complex systems is assumed to make the latter systems more controllable.

10. **Science of organization:** This challenge deals with the analysis and identification of common principles in the organization of complex systems such as companies, economic systems, societies, or natural systems. Among other things, a balanced state of these organization is of interest as well as events that throw them out of balance.

11. **Application areas:** The transformation of relevant research results into practice is essential. Application-specific solutions complemented with e. g. simulators or demonstrators are required, for instance in areas such as automotive, transportation systems, manufacturing automation, mechatronics, or power management.

Due to the overlaps as well as the vague descriptions and boundaries of these challenges – which mainly result from the individual interests of the variety of authors of [BBF+08] – many links exist between the problems and challenges tackled by this thesis and the challenges of the OC research area, in particular links to the challenges 1, 2, 3, 5, 7, 10, and 11.

2.5.2.2 Further Academic Initiatives

Beside OC there exist a few more initiatives driven by academia that tackle the management of future IT complexity. The research umbrella Autonomic Communication [Aut04] focuses in particular on the improvement of the ability of networks and services to cope with unpredicted changes. The latter include changes in topology, load, task, the physical and logical characteristics of the networks that can be accessed. In contrast to the pervasive perspective of OC, with focus on large collections of devices or systems, as well as the business perspective of AC, with focus rather on servers and databases, autonomic communication generally refers to

research thrusts involved in a deep foundational rethinking of communication, networking, and distributed computing paradigms to face the increasing complexities in these domains. Hence, the vision of autonomic communication research is that of a networked world in which networks and associated devices and services will be able to work in a totally unsupervised manner, able to self-configure, self-monitor, self-adapt, and self-heal (cf. [DDF+06]).

Amorpheus Computing [AAC+00] aims on assembling systems incorporating vast numbers of information processing units, such as micro-sensors, actuators, or communication devices, that show a coherent global behavior. It thereby allows for systems that comprise unreliable processing units interconnected in unknown ways. The targeted processing units are in contrast to other initiatives very small, for instance micro-fabricated particles that communicate via short-range radio or even bio-engineered cells that communicate by chemical means, e. g. wave propagation. An illustrative example application is a smart paint that senses and reports on wind loads on bridges it coats or monitors their structural integrity. Similar to OC, the challenge hence also amounts to how to engineer a pre-specified, coherent behavior from the cooperation of vast numbers of system elements.

2.6 Conclusion

In this chapter we have presented the necessary background knowledge for this thesis. More specifically, we have investigated the terms self-management, self-adaptation, as well as self-organization and emergence and have pointed out their conceptual relations, similarities, and differences more clearly. Based on that we have provided a definition of self-organizing emergent systems and presented MASs as technology to model and realize the former class of systems. At the end we have surveyed two research areas related to the problems and challenges tackled by this thesis.

Due to these investigations it becomes evident that the meaning of the terms self-management, self-adaptation, and self-organization is not clear-cut but depends very often on the community they are used in. According to the conceptual descriptions above, a self-managing system can be considered as a self-adaptive system as well as a self-organizing system, however, a self-adaptive system can be considered as a (weakly) self-organizing system too, whereas a self-organizing system is in general adaptive but not always self-adaptive. Thus, it is no wonder that in the AC as well as OC community the corresponding self-managing systems are often referred to as being self-organizing, even though conceptually self-adaptive was meant.

Anyway, our approach for efficient decentralized coordination (see Chapter 4, later) is based on the understanding of self-organization as presented in Section 2.3, whereas our approach for the efficiency improvement at runtime (see Chapter 6, later) is based on the understanding of self-adaptation as presented in Section 2.2. However, as one will see by the combination of these two approaches, the same system may be called strongly self-organizing as well as self-adaptive, subject to the time of consideration. This time aspect is not covered by any terminology yet.

Part II

Design Phase

Chapter 3

Designing Self-Organizing Emergent Systems

While the last chapter has provided the basics for this thesis in general and for self-organizing emergent (multi-agent) systems in particular, this chapter provides the background for and state of the art in *designing* self-organizing emergent systems. This will reveal in more detail, why the design process today is too complex, time-consuming, and costly (cf. *Problem 1*). The background for and state of the art in *operating* self-organizing emergent systems will be presented in Chapter 5.

For the design of self-organizing emergent MASs, two aspects play an important role: interaction and coordination. If the agents are not able to interact with each other, no global behavior will ever emerge. Moreover, to guide the global behavior in order to achieve the required system requirements, the appropriate, decentralized coordination of the agents' interactions is today acknowledged as a key issue for the design of self-organizing emergent MASs. Consequently, Section 3.1 provides the corresponding background by explaining different viewpoints on the coordination of interacting agents. By focusing on coordination models and languages, Section 3.2 then provides a comprehensive survey on the state of the art in coordinating computer systems in general and MASs in particular. Subsequently, Section 3.3 focuses on models and mechanisms for decentralized coordination, providing the state of the art in coordinating self-organizing emergent MASs. Section 3.4 subsequently examines the design process of self-organizing emergent MASs in the context of existing engineering methodologies. Finally, Section 3.5 concludes this chapter.

3.1 Subjective vs. Objective Coordination

Without doubt, everyone has an intuitive sense of the term *coordination*. When we watch a winning soccer team or aircrafts arriving and departing at an airport, we can notice obviously good coordination. However, we sometimes notice good coordination most clearly when it is lacking, e. g. when we spend minutes waiting on only one of a dozen elevators in a huge building, when we spend almost hours in an aircraft on a runway waiting for a starting slot, or spend even days waiting for our baggage that did not make it at an intermediate stop.

These examples indicate that the concept of coordination is not limited to MASs or even computer science. In sociology, researchers observe the behavior of groups

of people, and try to identify particular coordination mechanisms and explain how and why they emerge. In economy, researchers are concerned with the structure and dynamics of the market as a particular coordination mechanism and attempt to build coordination market models to predict its behavior. In biology, researchers observe societies of simple animals demonstrating coordination without central individuals and attempt to build biologically-inspired coordination mechanisms useful to other scientific disciplines. In organizational theory, the emphasis is on predicting future behavior and performance of an organization, assuming the validity of a certain coordination mechanism. But coordination also has its role in many other disciplines, such as social sciences, anthropology, linguistics, law, or political science (cf. [MC94]).

A simple but widely accepted definition of coordination in general was given by Malone and Crowston [MC94]: *"Coordination is managing dependencies between activities."* This definition is consistent with the simple intuition that, if there is no interdependence, there is nothing to coordinate. Consequently, the definition is not restricted to computer science, but also applies to many other disciplines. The fairly inclusive sense of the definition also allows for specialized forms of coordination, such as cooperation, collaboration, and competition. *Cooperation* usually implies shared goals among different actors. *Collaboration* often connotes peers working together on an intellectual endeavor. *Competition* usually implies that one actor's gains are another's losses. However, all these terms describe different approaches to managing dependencies between activities (cf. [MC94]).

In computer science, research on coordination has developed along two basically separated fields, namely DAI and SE, respectively. In DAI, for a long time coordination was interpreted as an individual, psychological activity, performed by a agent trying to achieve its own goals in the context of a MAS. Here, the agents were seen as the *coordinating* elements. In SE, coordination was basically regarded as normative activity performed by some part of a multi-component system on behalf of the system's designer – typically, by a coordination medium provided by an infrastructure. Here, the components were seen as the *coordinated* elements. Whereas the first approach seems to better suit systems whose components exhibit a high degree of autonomy, the second approach often disregarded any capability of the components in terms of autonomy or deliberation (cf. [ORVR04]).

Schumacher [Sch01] was the first attempting to combine these different viewpoints. He argued, that basically there are two ways to look at interactions between agents: from the inside (agent-oriented viewpoint) and from the outside (MAS-oriented viewpoint) of the interacting agents. Due to the definition of coordination, these two different viewpoints result in two different ways of coordinating agents in a MAS (cf. [Sch01, OO03]):

- **Subjective coordination** refers to the coordination from the agent-oriented viewpoint and deals with managing subjective dependencies between agents, i. e. *intra-agent dependencies* towards other agents. It affects the way in which individual agents behave and interact, focusing on the tasks, goals, plans, and

actions of an agent. The interaction space of an agent roughly amounts to the observable behavior of other agents and the evolution of the environment over time, filtered and interpreted according to the agent's perception and understanding. In other words, the agent monitors all interactions that are perceivable and relevant to it, as well as their evolution over time, and devises actions that could bring the overall state of the MAS (or the world) to better coincide with one of its own goals or the goals of the agency.

- **Objective coordination** refers to the coordination from the MAS-oriented viewpoint and deals with managing objective dependencies between agents, i. e. *inter-agent dependencies* external to the agents. The latter include e. g. the configuration of the system in terms of the basic interaction means, agent generation/destruction, and organization of the environment. Objective coordination acts directly on the dependencies in an environment and affects the way in which interaction amongst the agents and the environment is enabled and ruled and is hence mainly concerned with the organization of the world of a MAS. The interaction space is given by the environment of a MAS, the observable behavior of all the agents, and by all their interaction histories.

Roughly speaking, subjective coordination affects the way in which individual agents behave and interact, focusing on the behavior of agents as (social) individuals immersed in a MAS. In contrast objective coordination affects the way in which interaction amongst the agent and the environment is enabled and ruled, focusing more on the behavior of a MAS as a whole. Not differentiating subjective and objective coordination leads to MASs that resolve objective coordination with subjective coordination means, i. e. by using intra-agent aspects for describing system configurations. This mixture is typically present in MASs composed of deliberative agents using ACLs in order to communicate (cf. [Sch01]). For a more detailed differentiation between subjective and objective coordination see [OO03], which use a simple example scenario from the blocks world for better illustration.

Due to the differentiation between subjective and objective coordination, the role of the environment of a MAS began to change. Traditionally, the environment of a MAS has been very often associated with the infrastructure for a MAS (see e. g. [HS99b]). The infrastructure is essential for a MAS, because it offers the MAS important functionalities, such as the communication infrastructure, network topology, available physical resources, naming, life-cycle-management, etc. (cf. [OOR04]). In other words, the environment has been viewed as a *passive entity* of a MAS constituting the deployment context where agents are immersed in. With the notion of objective coordination, however, the environment of a MAS became more than a sole infrastructure, as it had not only to enable interactions but also to rule it. Thus, it became the main place for objective coordination. The role of the environment became even more crucial, when it came about the modeling of self-organizing emergent MASs. The observation of self-organizing processes in natural systems, for instance pheromone diffusion and evaporation in stigmergic systems

([BDT99, CDF+01]), as well as experience with respective artificial systems (see e. g. [SMPB05, GVO07]) indicated that there is a class of self-organizing processes that is best modeled as a part of the environment rather than expressed in terms of agents [GVCO08]. Hence, in recent years it became commonly acknowledged that the environment is also a true design dimension of MAS applications [WPM+05]. The environment can encapsulate a significant portion of the system's complexity, in terms of services, mechanisms, and responsibilities that the agents can fruitfully be freed of [PMS+07]. Consequently, Weyns et al. [WOO07] define the environment of a MAS as *"a first-class abstraction that provides the surrounding conditions for agents to exist and that mediates both the interaction among agents and the access to resources."* The environment hence became an *active entity* of a MAS that regulates particular activities of the system in addition to the mediation of interactions, which enables agents to exploit the environment to coordinate their behavior.

To clarify the difference between the concept of an active environment and the infrastructure on which a MAS is deployed, Weyns et al. [WVH06] proposed a logical three-layer model for MAS that considers agents as well as the environment as first-order abstractions. Viroli et al. [VHR+07] modified this model slightly to emphasize the role of environment infrastructures superiorly (see Fig. 3.1). The model is made up of three layers: the *physical infrastructure* (i. e. the hardware deployment context), the *execution platform* (i. e. the software layer over which the MAS runs), and the *MAS application*. Rectangles represent software and hardware tiers of the system at different levels. Circles represent agents, boxes represent environment abstractions. Solid arrows from agents to environment abstractions represent actions, dashed arrows in the opposite direction represent perceptions. Arrows between agents represent direct agent communications, while arrows between environment abstractions represent intra-environment interactions. Vertical lines represent the infrastructure supporting a concept at the MAS application level.

The physical infrastructure is divided into two parts: the *physical world*, which refers to the physical parts of the MAS, and *hardware & network*, which contains hosts, processors, network infrastructure, etc. The execution platform runs on top of the physical infrastructure, which is generally divided into two parts again: the *software deployment context*, which includes operating systems, virtual machines, and other (non MAS-related) standard middlewares, and the *MAS middleware layer*. In general, there can be two kinds of MAS middlewares: (1) an infrastructure for agents providing agent life-cycle, management, and often some other core services like direct communication (e. g. JADE [BPR01], JACK [NR01], FIPA platform [FIP02a], Retsina [SPVG03], Living Systems [Whi09], etc.), and (2) one (or more) environment infrastructures, each providing some class of environment abstractions to agents for creation, access and manipulation (e.g. TuCSoN [OZ98], AMELI [ERRAA04], etc). Environment abstractions are entities that an agent might perceive and interact with in order to achieve individual or social goals. From the design viewpoint, environment abstractions are seen as loci where the designer can enforce rules, norms, and functions, regulating the agent social behavior [VHR+07].

The part of the MAS middleware software layer dealing with the environment is

therefore understood as an infrastructure providing some class of environment abstractions at run-time. It is worth noting that agent and environment infrastructures cannot be completely isolated. An intersection should exist that handles agent to environment interactions, that is, agent actions and perceptions over environment abstractions. On top of these layers, the *MAS application* is formed by *application agents* living over the agent infrastructure, and environment abstractions in the *application environment* provided by the environment infrastructures. Interactions between agents and environment abstractions (actions/perceptions) necessarily cross the agents and environment boxes. But interactions between agents respectively between environment abstractions can occur as well (cf. [WVH06, VHR+07]).

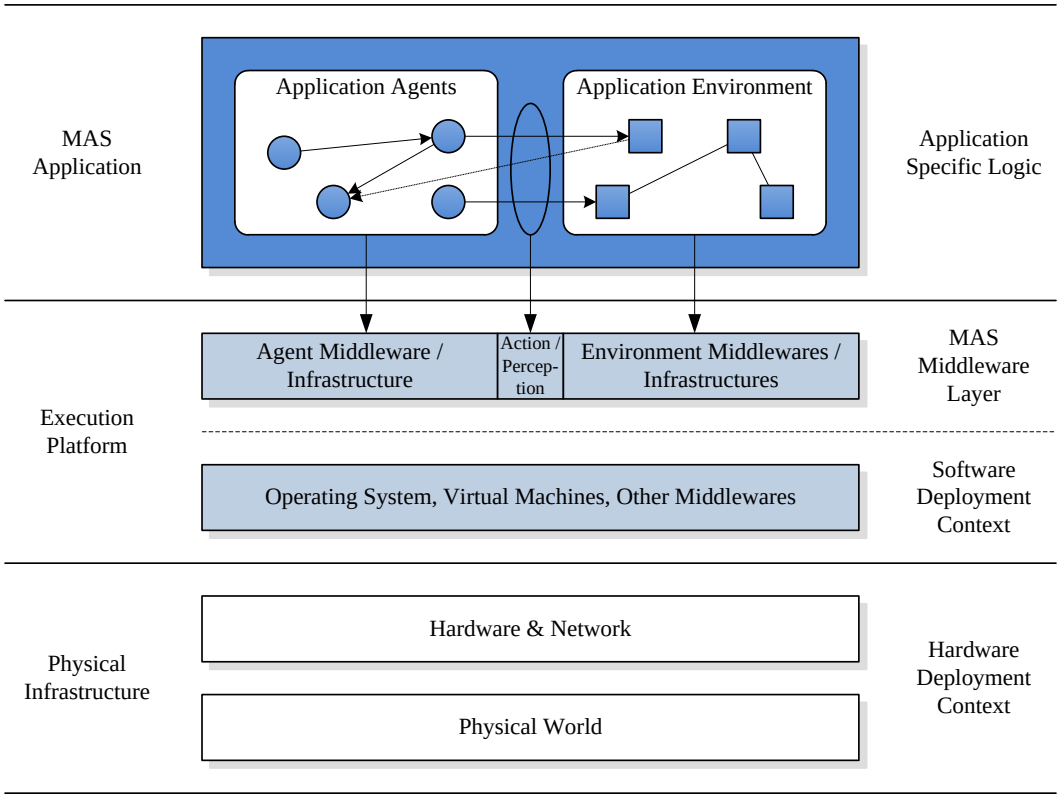


Figure 3.1: Three layer model for multi-agent systems (adapted from [VHR+07])

In the case of a self-organizing emergent MAS, an *active environment* (along with its corresponding infrastructure) is in charge of facilitating the adaptive self-organization of agents that act according to a matching decentralized coordination mechanism. The environment therefore has to provide suitable means that support the following basic requirements (cf. [MZ06, KG06, DDF+06, GVCO08]):

- **Indirectness:** For the coordination of agents in open and dynamic systems, in which agents may leave or join the system at any time and any place,

indirect interactions among the agents, e. g. as in the case of stigmergy in nature, are much more suited. Environments facilitating indirect interactions hence uncouple the interacting agents and free them from the need for directly knowing each other to interact, which promotes spontaneous interactions that are essential to any self-organizing process.

- **Context-awareness:** In order to facilitate spontaneous indirect interactions to take place among the agents in an adaptive manner, the agents must affect the surrounding environment by their actions in a way that can be perceived and exploited by other agents for their application purposes. In other words, indirect interactions among the agents require the capability of affecting and perceiving context information.

- **Locality:** For the purpose of effective and scalable coordination in large-scale systems, the environment should also promote locality in both the interactions and the acquisition of contextual information. Any approach that requires global interactions in systems comprising large numbers of agents is doomed to fail. Scalability and ease of management can only be properly supported by an environment, in which most interactions between the agents occur at the local level.

As a result, the environment as the coordination medium for indirect interactions provides a kind of "service", which in turn is used by the coordination mechanism that is engaged to ensure subjective coordination among the agents. In other words, by choosing a certain coordination medium, a coordinated agent or MAS implicitly commits to both the coordination model endorsed by the environment/infrastructure providing the medium and the specific coordination rules embodied by the medium (cf. [VO06]). Consequently, subjective coordination is based on and supposes the existence of objective coordination, because a coordination mechanism must have access to the mechanisms of the coordination medium [SO06]. Vice versa, objective coordination will not give rise to a self-organizing behavior of agents of itself without suitable mechanisms for subjective coordination. Thus, it is important that subjective and objective coordination complement each other and that any attempt to put subjective and objective coordination altogether should aim at providing a uniform conceptual framework [ORVR04].

While this section therewith has provided the necessary background for the coordination of agents in self-organizing emergent MASs, the next section focuses more on objective coordination, which mainly amounts to various existing coordination models and languages, whereas Section 3.3 but one focuses more on subjective coordination, which mainly amounts to existing models and mechanism for the decentralized coordination of agents.

3.2 Coordination Models and Languages

In order to affect the way in which interaction amongst agents and their environment is enabled and ruled, objective coordination requires a well-founded *coordination model*. A coordination model comprises three ingredients (cf. [Weg96, Cia96]):

- **Coordination elements (coordinables):** the elements whose interactions are ruled by the model, which can be agents, as of interest here, but in general also processes, threads, concurrent objects, and even users.
- **Coordination media:** the media used to coordinate the elements, i. e. the abstractions enabling the interactions, e. g. semaphores, monitors, channels, or more complex media such as tuple spaces, blackboards, pipelines, etc.
- **Coordination laws:** the semantic framework the model adheres to, i. e. a number of laws that define the behavior of the coordination media in response to interaction events.

In general, a coordination model supports the clear separation between the computation and the coordination aspects of programming, which is necessary to master the complexity of large applications, to enhance software re-usability, and to ease global analysis [GC92]. The importance of separating these aspects may also be summarized by the statement that *"interaction is more important than algorithms"* [Weg97] and is advocated by the following slogan [GC92]:

$$\text{programming} = \text{computation} + \text{coordination}$$

Consequently, a complete programming model can be build out of two separate models: the *computation model* and the coordination model. Whereas the computation model allows programmers to build a single computational activity, i. e. a single-threaded, step-at-a-time computation, the coordination model serves as the glue that binds separate activities into an ensemble [GC92]. In other words, a coordination model provides a framework in which the interaction of active and computational independent elements can be expressed [Cia96].

Whereas an ordinary *computation language* embodies some computation model, a *coordination language* embodies a coordination model. The latter can be viewed as the *"linguistic embodiment of a coordination model"* [GC92] that provides operations to create computational activities and to support communication among them. It offers syntactical means with which a coordination model can be used for implementing an application. A coordination language consists of a small number of mechanisms for communication and process management that are orthogonal to the mechanisms used to describe the internal computations of each (sequential) process. It offers facilities for controlling synchronization, communication, as well as creation and termination of computational activities.

Over the last five decades, in more detail since the debut of the first distributed systems – IBM's SAGE computers and shortly after IBM's SABRE reservation system –, there have been huge strands of work on diverse coordination models and languages for different technologies, mechanisms, and applications. Apparently, not all coordination models since the 1960s have been developed for agents in the first instance, even though many coordination models for MASs and later self-organizing emergent MASs are based on earlier concepts. Thus, a huge spectrum of coordination models has emerged, ranging from models that have not been developed to support self-organizing and emergent phenomena in the first instance up to models that support at least a few if not all principles of self-organization and emergence, in which elements gather relevant information on their own and decide for themselves what actions and interactions to perform. Consequently, a taxonomy providing a clear differentiation between coordination models for concurrent and distributed systems in general and self-organizing emergent systems in particular is hard to establish. For instance, Ossowski and Menezes [OM06] propose the term *dependent coordination* models to distinguish from so-called *emergent coordination* models. However, the proposed characteristics of dependent and emergent coordination models are very restrictive and anything but clear-cut. Viroli et al. [VCO09] in contrast use the term *self-organizing coordination* instead of emergent coordination, however, with very different characteristics compared to Ossowski and Menezes.

Thus, instead of using or proposing a taxonomy, in this section we present existing coordination models and languages that are the current standard for conventional distributed systems. These models mainly rely on predictable coordination rules whose impact on system interactions is known and fully re-producible. The coordination models and mechanisms for decentralized coordination presented in the next section are, however, to some extent based on these conventional models. But even for conventional coordination models up to today different, partly overlapping taxonomies have been proposed that aim to structure the variety of these models. Although none of these taxonomies provides an unambiguous classification that covers all existing coordination approaches, we will use two of them to describe and explain the main characteristics and differences between coordination models.

3.2.1 Data-driven vs. Control-driven Coordination

Papadopoulos and Arbab [PA98] proposed the first and still most cited taxonomy of coordination models and languages with focus on concurrent and distributed computations[1]. They distinguish between data-driven and control-driven (or task- or process-oriented) coordination models, even though they stress the point that the data- vs. control-driven separation is also by no means a clear-cut one.

[1] Arbab [Arb98] also proposed another classification by distinguishing coordination models and languages between endogenous or exogenous. Endogenous models and languages provide primitives that must be incorporated *within* a computation for its coordination, whereas exogenous models and languages support coordination of entities from *without*. However, this taxonomy did not prevail.

3.2.1.1 Data-driven Coordination

In *data-driven coordination* models, *"the state of a program is defined in terms of both the values of the data being received or sent and the actual configuration of the coordinated components"* [PA98]. This means that coordination entities are in charge of handling the data and coordinating themselves with the other entities. In almost all coordination models belonging to this category, the coordination media is represented by a shared data space [RC90], which is a common, content-addressable data structure shared between the coordination entities. All involved entities can communicate among themselves only indirectly via this coordination medium. They can post or broadcast information into the medium and also they can retrieve information from the medium either by actually removing this information out of the shared medium or merely taking a copy of it.

The Linda coordination model respectively language [Gel85, Gel89] is a classic example in this category. It is based on the so-called *generative communication* paradigm: if two processes wish to exchange some data, then the sender generates a new data object (referred to as a *tuple*) and places it in some shared data space (known as a *tuple space*) from which the receiver can retrieve it. This paradigm decouples processes in both space and time. Since its proposal, Linda has spawned and inspired tens of variations. They extend Linda for instance by introducing multiple data space and meta-level-control rules (e. g. Bauhaus Linda [CGZ95], Bonita [RW97], Objective Linda [Kie96], PoliS [Cia91], Shared Prolog [BC91], Ariadne/HOPLa [FBG96], and Sonia [Ban96]), by using logical operators (e. g. LogOp [SM02]), by addressing open-distributed systems (e. g. LAURA [Tol96]), middleware-based environments (e. g. Jada [CR97] and SHADE [CCR96], which combine Java with Linda for the Web) or mobility (e. g. KLAIM [dNFP98]).

Another model of this data-driven category is GAMMA [BM90], which is based on the chemical reaction metaphor [BFM01]. GAMMA is one of the first proposals of models based on *multiset rewriting*. This coordination medium is composed of multisets, i. e. sets whose elements can have multiple copies. A coordination model is then concerned with the manipulation of these multisets, mostly defining rewrite rules. Since its proposal, GAMMA has also spawned several extensions, e. g. CHAM [BM93], IAM [ACP93], LO [AP91] and its extension COOLL [CC96].

Further models of this category are Concurrent Constraint Programming [Sar93], MESSENGERS [FBDM96], and Opus [CHM+97], which all have weak similarities to the above models. More recent members of this category are PageSpace [CTV+98], JavaSpaces [FHA99] (a component of the Jini reference architecture [Wal99]), and EventHeap [JF02]. Even XMIDDLE [MCE01], PeerSpaces [BMMZ03], and Lime [MPR06] belong to the category of data-driven models, although there the shared data space is carried by mobile elements and dynamically merged with each other.

3.2.1.2 Control-Driven Coordination

Within *control-driven coordination* models, *"the state of the computation at any moment in time is defined in terms of only the coordinated patterns that the processes involved in some computation adhere to"* [PA98]. Therefore an application is centered on the processing or flow of control. In control-driven coordination models, the coordination media is represented by a set of input/output communication ports linking the coordination entities and enabling their interactions. Connections between the ports are mostly of a channel nature. Contrary to the data-driven category where the coordination entity "sees" the manipulated data and directly handles and examines data values, here the entities are treated as black boxes where data handled within an entity is of no concern to its environment. The coordinated framework evolves by means of observing state changes in processes and, possibly, broadcast of events.

A typical control-driven model that defines a family of models is IWIM [Arb96]. It defines processes, ports, channels, and events. A process is considered as a black box. To communicate, a process writes and reads units (data) on its ports, which are named openings in the bounding wall of the process. Thereby the process is not aware to whom information is transmitted to, as the communication is anonymous. The established connections between ports are of a channel nature. A channel is considered to be unidirectional, connecting a producer process (source side) to a consumer process (sink side), and reliable, assuring that units are transferred without loss, error, or duplication. Reading on a port blocks the corresponding port until a unit is available. Writing on a port blocks as long as the port is not connected. Events are used for information exchange. They usually indicate process state information to other processes. An event is therefore a signal broadcast in the environment, causing an event occurrence, composed of the identity of the event and of its producer. Every process in an environment may capture an event occurrence and react to it. However an event is caught only by interested processes. A coordination language that is based on IWIM is e. g. MANIFOLD [AHS93]. Other coordination languages of the control-driven category are e. g. ConCoord [Hol96], Durra [BWD+93], Programmer's Playground [GSM+95], RAPIDE [SDK+95], Darwin [MDK93], TOOLBUS [KB96], Contextual Coordination Model [BB97], CoLa [HAK94], and Reo [Arb04].

3.2.1.3 Hybrid Coordination

Whereas the data-driven category tends to be used mostly for parallelizing computational problems, the control-driven category tends to be used primarily for modeling systems (cf. [PA98]). However, Ciancarini et al. [COZ99] argue that data-driven models seem to better suit open systems, where a number of possibly a-priori unknown and autonomous entities have to cooperate, as typical in the case of MASs. A typical problem of data-driven coordinated systems is the built-in and fixed behavior of the shared data space used as the coordination medium: neither new

communication primitives can be added, nor can new behavior be defined in response to standard communication events. To address this deficiency, Ciancarini et al. [COZ00] propose a third category of coordination models, *hybrid coordination* models, which are meant to combine the cleanness and elegance of data-driven models with the flexibility and power of control-driven ones.

A first group of work integrates the event mechanism of control-driven models into shared data space models. Examples are ACLT [ODN95], TuCSoN [OZ98], and MARS [CLZ99]. Further hybrid models that combine control-driven elements with a data-driven models are T Spaces [Pet98], Law-Governed Linda [ML95], and IWIM-Linda [PA97]. All of these models center around the idea of *programmable coordination media* [DNO97], i. e. when a communication operation is executed, a reaction catching the event produced atomically executes a sequence of operations, which usually have access to both the space and the information associated with the event. The ECM coordination model [SCH99], along with its instantiating coordination languages STL, STL++, and Agent & Co., is also a hybrid model, because it integrates shared data space functionalities in a process-oriented view (cf. [Sch01]).

3.2.2 Spatial and/or Temporal Coupled/Uncoupled Coordination

Around the turn of the millennium, agent technology and the Internet have attracted widespread interest, which put forth new coordination models as well as taxonomies. Papadopoulos [Pap01] surveys some of the most common models and technologies that offer mechanisms for the coordination of Internet agents, which include several of the above models and languages again. Bocchi and Ciancarini [BC03] also present a review of many of the aforementioned coordination models and languages, in particular in the context of web services and the semantic web.

Cabri et al. [CLZ00], however, proposed a new taxonomy for coordination models, which is based on the degrees of spatial and temporal coupling induced by a coordination model. Although the focus of this taxonomy was primarily on mobile agents for Internet applications, it also serves as a taxonomy for distributed systems in general, as it became adapted by [TvS06]. According to this taxonomy, spatially coupled coordination models require that the interacting entities share a common name space. In turn, spatially uncoupled models enforce anonymous interactions. Temporally coupled coordination models imply synchronization of the entities involved. In turn, temporally uncoupled coordination models achieve asynchronous interactions. Cabri et al. thus derive four main coordination model categories from the combinations of these characteristics: direct, blackboard-based, meeting-oriented, and Linda-like coordination (see Figure 3.2). Although this taxonomy provides no clear-cut categorization, it is more specific than e. g. the taxonomy proposed by Keil and Goldin [KG03], who only distinguish *direct coordination* models vs. *indirect coordination* models.

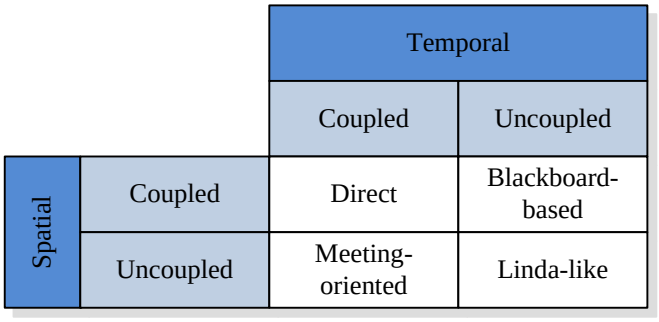


Figure 3.2: Taxonomy of coordination models distinguishing spatial and/or temporal coupled/uncoupled models (adapted from [CLZ00])

3.2.2.1 Direct Coordination

Direct coordination models usually imply spatial and temporal coupling of the agents. The agents start a communication by explicitly naming the partners involved. In the case of inter-agent coordination, two agents therefore must talk the same language and thus have to agree on a communication protocol. This can be handled on the one hand by client-server approaches using Remote Procedure Call (RPC)-like or Remote Method Invocation (RMI)-like primitives (see e. g. the Jini middleware [Wal99]) and on the other hand by asynchronous message passing (see e. g. UPnP [UPn09] or JADE [BPR01]). The majority of the Java-based mobile agent systems [SARDS01], particularly the most famous ones such as Aglets [LO98], Voyager, [Gla98], Ajanta [TKA+02], and GrassHopper [IKV98], belong to this category. Also the Actors coordination model [Agh86] is based on direct coordination.

3.2.2.2 Meeting-oriented Coordination

In *meeting-oriented coordination* models, the agents interact in the context of meetings without needing to explicitly name the partners involved, so they usually imply spatial uncoupling of the agents but still a temporal coupling. Agents join either explicitly or implicitly known meeting points; afterward, they can communicate and synchronize with the other agents that participate in such meetings. However, meeting-oriented coordination models cannot achieve the anonymity of full spatial uncoupling, as the agents must share at least the common knowledge of the meeting names. A typical example for meeting-oriented coordination is the Ara [PS97] implementation. A sophisticated form of meeting-oriented coordination was introduced by the concept of event-based communication and synchronization or publish/subscribe coordination (see below), e. g. implemented by Mole [BHRS98].

3.2.2.3 Blackboard-based Coordination

In *blackboard-based coordination* models, the agents interact via shared data spaces, using them as common intermediary repositories to store and retrieve messages.

These models exploit the classic blackboard architecture [HR85]. In this sense, interactions are fully temporally uncoupled, but, because agents must agree on a common message identifier to communicate and exchange data via a blackboard, they remain spatially coupled. This model is used by Ambit [CG98] and ffMAIN [DLD97]. However, there also exist some blackboard-based systems that do not have a spatial coupling.

3.2.2.4 Linda-like Coordination

Linda-like coordination models use local tuple spaces as message containers similar to blackboards. The system organizes information in tuples and retrieves it using associative pattern-matching (see above). This approach enforces full uncoupling, requiring neither temporal nor spatial agreement. Examples for Linda-like coordination are e. g. again PageSpace, TuCSoN, and MARS.

3.2.2.5 Publish/Subscribe Coordination

Although not explicitly mentioned in the taxonomy of Cabri, another important family of coordination models are *publish/subscribe (event-based) coordination* models. Eugster et al. [EFGK03] provide a survey on this family, however in a very broad sense. Based on their definition this family can be classified into the same category as Linda-like coordination, i. e. temporally and spatially uncoupled. However, there also exist subscription models, which presume a spatial and/or temporal coupling, as e. g. some models based on meeting-oriented coordination, as mentioned. In general, in publish/subscribe models agents coordinate through asynchronous publication and notification of events. In order to be notified by an event, an agent has to subscribe to an event type; if an event of the subscribed event type occurs, the agent will (potentially immediately) be notified. Publish/subscribe models enable what can be regarded as active tuple spaces. Rather than expecting processes to continuously poll for a given tuple in a tuple space, publish and subscribe strategies enable processes to subscribe to specific patterns of messages or tuples, and to publish tuples with the expectation that all currently subscribed processes will receive a copy of the tuple.

Depending on the subscription model, publish/subscribe systems can be classified into topic-based (or channel-based respectively subject-based) models, e. g. Scribe [CDKR02], content-based models, e. g. Siena [CRW01] or REBECA [Müh02], or type-based models, e. g. Hermes [PB02]. Further models that belong to one or more of these classes include Elvin [SA97], Gryphon [SBC+98], Le Subscribe [PFL+00], Jedi [CNF01], WebFilter [PFJ+01], NaradaBrokering [PF03], IndiQoS [CAR05], ECho [ESB06], GREEN [SBC05], REDS [CP06], and Cobra [RMP+07].

Whereas all these models are mainly research prototypes, commercial systems and standards supporting the publish/subscribe model have appeared, too. Some of the most popular systems include IBM WebSphere MQ (IBM MQSeries) [IBM09], TIBCO Rendezvous [TIB09], and the ORACLE WebLogic Server [ORA09]. But

also industry standards related to publish/subscribe systems exist, such as the Java Message Service (JMS) [Sun02a], the CORBA Event Service [OMG04a], the CORBA Notification Service [OMG04b], and the OMG Data Distribution Service [OMG07].

Note, shared data space models can be reduced to the publish/subscribe architectures, whereas the vice versa holds only if a specific, global coordination operation is provided among the shared data space operations (cf. [BZ01]).

3.2.3 Bottom Line

According to the requirements for the environment of a self-organizing emergent MASs listed at the end of Section 3.1, conventional coordination models that are based on direct message-passing, client/server models, or shared data spaces are inadequate to deal with the requirements for decentralized coordination in self-organizing emergent MAS systems (cf. *Challenge 1*). There are several reasons for this inadequacy (cf. [DDF+06, GVO07]):

1. They do not support indirect interactions between agents nor a strong notion of the environment.

2. They do not account for context-awareness and meaningful interactions, which forces system elements to operate in a kind of "void", where the only things that exist are the other system elements.

3. They rely on static assumptions and a priori knowledge about a system, i. e. a spatial-temporal coupling and referential awareness, which forces system elements to operate in the same system at the same period as well as to know each other.

As a consequence, most conventional coordination mechanisms for MASs developed by DAI research that are based on these conventional coordination models, such as organizational structuring, multi-agent planning, negotiation, or contracting (see [Jen93, Jen96, NLJ97, Oss99]), become inadequate for self-organizing emergent MAS systems, too. These coordination mechanisms moreover typically consider coordination as *"the process by which an agent reasons about its local action and the (anticipated) actions of others to try and ensure the community acts in a coherent manner"* [Jen96]. Thus, the coordination of agents is already decided in early design time and does not account for open and dynamic systems. Although some dynamic agent activity adjustments can be described, e. g. dynamic role assignment [OPBS04], changing coordination mechanisms at run-time enforces considerable effort (cf. e. g. [ETJ04]).

However, there exist two conventional coordination models that are able to deal at least with some of the requirements for decentralized coordination: publish/-subscribe models as well as Linda-like models. Publish/subscribe models already support certain flexible and uncoupled interactions, which are suitable for future dynamic scenarios. Some modern approaches, e. g. Siena, even provide distributed

event-dispatching services in the context of mobile ad hoc networks, in which mobile nodes engage a distributed algorithm to self-organize event-dispatching routes and to maintain such routes despite network dynamics. However, publish/subscribe models will support only a small number of decentralized coordination mechanisms effectively, while several phenomena of self-organization can hardly be mapped in terms of publish/subscribe patterns. Due to the coordination process in most Linda-like coordination models, the latter achieve already some forms of context-awareness and the possibility of interacting with unknown processes/agents in an uncoupled way. In particular more recent approaches, e. g. Lime, exploit distributed or transiently-shared tuple spaces, which forms a basis for programming interactions in dynamic scenarios. However, the semantics of tuple-space interactions require extensive use of synchronization, while these models can not be used to effectively program and enforce phenomena of self-organization and emergence without further ado, in particular because the environment lacks any form of structuring (cf. [DDF+06]).

3.3 Decentralized Coordination

Due to these inadequacies, models and mechanisms for decentralized coordination support principles of self-organization and emergence and hence are suitable for application scenarios, where dynamic disturbances, unpredictability, openness, and large-scaled systems are key properties. In most of these approaches there is no 'direct correspondence' between the purpose of local interaction, and the global functionality of coordination within a system. Thus, decentralized coordination as understood here is very similar to the description of self-organizing coordination proposed in [VCO09]: *"Self-organizing coordination is the management of system interactions featuring self-organizing properties, namely, where interactions are local, and global desired effects of coordination appear by emergence."* The addressed *global desired effects* thereby correspond to the functionality a self-organizing emergent solution is designed for to fulfill. Depending on the application scenario, categories of such functionalities are very often, but not exclusively, the following ones (cf. [DH07a]):

- **Resource allocation:** Certain application scenarios require that the self-organizing emergent solution autonomously manages possibly limited resources, which can be tasks, power, goods, bandwidth, space, (CPU) time, devices, machines, etc., by deciding which system element can use a resource or is assigned to a resource.

- **Group formation:** Some application scenarios require groups or teams to be formed autonomously. As such, these groups or teams can be adapted and restructured based on specific changes that occur in the environment or the system itself. Even clustering of items or data can be considered as a kind of group formation.

- **Role-based organizations:** Several application scenarios require to enforce so-called organizations, which in MASs comprise roles and their interrelations that impose a formal structure on the system. Roles, e. g. master roles and slave roles, thereby cluster certain types of behavior into meaningful units that contribute to the groups overall goals. Role interrelations, e. g. hierarchical or class-membership relations, provide communication paths among agents. Together with certain interdependencies, e. g. temporal or resource dependency relations, roles and their interrelations can be exploited for effective agent coordination and communication.

- **Self-protection:** A growing number of application scenarios require from a self-organizing emergent system to detect, identify, and protect itself against various types of attacks (see also Section 2.1). In the first instance, this is to maintain the overall system security and integrity. However, more than just responding to failures or running periodic checks for symptoms, self-organizing emergent systems need to remain on alert, anticipate threats, and take necessary action, which can even imply a coordinated group-defense.

- **Spatial structure:** Many application scenarios require that a certain spatial structure is constructed and maintained in the face of dynamic changes in the environment or the system itself. Thereby, two different types of spatial structure can be distinguished:

  - **Specific shapes:** Specific spatial shapes, e. g. certain topologies or paths, have to be constructed and maintained in a self-organizing manner.
  - **Spatial distribution:** A specific spatial distribution, e. g. an equal distribution of system elements in a certain region of the environment, has to be acquired and maintained in a self-organizing manner.

- **Information dissemination[2]:** Some application scenarios require new or updated information, e. g. on specific system elements or the environment, to be spread in order to be shared by all system elements. Upon the receiving of new or updated information, a system element can update its own knowledge on other system elements or the environment, whereby the elements may acquire almost global knowledge.

In regard to the design of self-organizing emergent solutions, two additional aspects play an important role for decentralized coordination:

- **Indirect interaction:** In order to support self-organizing and emergent phenomena best, it is required to support indirect interactions between agents as well as a strong notion of the environment. Indirect interaction is achieved through coordination media spread over the topological environment, enacting probabilistic and time-dependent coordination rules.

[2]This category was originally not covered by [DH07a].

- **Source of inspiration:** In order to support engineers in specifying new coordination mechanisms based on a specific decentralized coordination approach, the latter inherently has to provide an (idealistically inexhaustible) source of inspiration.

Because the family of decentralized coordination approaches is still in its growth, a precise classification has not been provided yet, even though considerable effort has been spend to compare and classify different coordination models and mechanisms (see e. g. [DH06, MMTZ06, SGK06, DH07b, PMS+07, DDF+06]). As a consequence, today it is also unclear, which coordination approach(es) will prove most suited for given problem domains, which naturally complicates the task for an engineer to identify and probably integrate the most suited approaches for his problem. Nonetheless, some main classes of decentralized coordination models and mechanisms can be distinguished.

3.3.1 Market-based Coordination

Inspired by economics, in market-based coordination [Bre01, DZKS06], computational systems are viewed as virtual marketplaces in which economic elements, such as agents or robots for instance, interact by buying and selling. Although decision-making by these elements is very often local, economic theory in most cases provides means to generate and predict macroscopic properties. The most common decentralized coordination mechanism used in market-based coordination approaches are auctions (see [Wol96, WWWMM01, HA07] for a more detailed overview). In an auction, a set of items is offered by an auctioneer in an announcement phase, and the participants can make an offer for these items by submitting bids to the auctioneer. Once all bids are received or a pre-specified deadline has passed, the auction is then cleared in the winner determination phase by the auctioneer who decides which items to award and to whom. Typically, the items for sale are tasks, roles, or resources (see e. g. [GJVG99]). The bid prices reflect the participants' costs or utilities associated with completing a task, satisfying a role, or utilizing a resource. In theory, incorporation of several rounds in which bids are generated, reviewed, and updated improves the negotiation outcome. Practically, several rounds are often not necessary as computational elements such as agents are able to compute the compromise they will eventually reach and settle for it. The best known auction type is the ascending-price open-cry or English auction, followed by the first-price sealed-bid or Dutch auction, whereas the most frequently used auction type is the second-price sealed-bid or Vickrey auction.

Similarly to auctions, the Contract Net Protocol (CNP) [Smi80] is an extensively used protocol in particular for task assignment. This market-based coordination mechanism, which is also included in a FIPA standard [FIP02c], is based on the notion of call for bids, where managers request for bids and the bidders bid to perform the task. In the first step, the manager sends a description of the task to perform to all the bidders. In the second step, the bidders draw up a proposal

based on the description of the task and send it to the manager. Finally, in the third step, after the manager has received a proposal from all the bidders or after a deadline has expired, the manager evaluates the received proposals and assigns the task to the best bidder. In the original specification of the CNP a bidder is not allowed to place a bid for a new task before its current task was finished. Since its initial introduction, the CNP has undergone a couple of improvements and extensions, e. g. the ability for a bidder to place bids for multiple unassigned tasks or to participate in cascading task assignments [KSF02], the ability of leveled commitment contracts (agreements between agents that can be withdrawn) [SL02] respectively decommitments [HP04], or the use of a pre-bidding phase before the definitive bidding phase for the reconsideration of commitments in the first phase [APS04].

Typically market-based coordination is used to achieve efficient *resource allocation*. However, the coordination approach may also support *spatial distribution* when considered in a network in which system elements have to find resources (CPU, disk storage) to complete an assigned task. High prices to purchase the resources necessary to complete a task may connote congestion, forcing the elements to choose other hosts with lower prices, which lets emerge an equal distribution. Usually, market-based coordination mechanisms are based on coordination models favoring direct interactions. However, the AMELI infrastructure [ERRAA04] for instance can be considered as an approach that enables *indirect interactions* for a subgroup of market-based coordination systems. AMELI is based on electronic institutions that represent normative systems. These institutions include scenes that agents can enter and thus provide a place where agents can meet and exchange messages in a regulated way, where prohibited interactions are disallowed by the infrastructure.

3.3.2 Gossip-based Coordination

An approach for decentralized coordination inspired by social human behavior is gossip-based or epidemic coordination [BJ08]. The basic idea is that at regular time intervals, each individual in a population exchanges information with a randomly selected individual from the population, followed by updating its local state based on the information exchange. This is very similar to gossiping in social networks. Such mechanisms allow for the aggregation of a global information inside a population through a periodic exchange and update of individual information among the members of the population. Initially, gossip-based coordination has been used for *information dissemination*, in more detail the propagation of updates among replicas of a database [DGH+87]. More recently, a gossip-based protocol has been used to achieve *spatial shapes* in the topology management in P2P systems [JB05], achieving high robustness, scalability, flexibility, and simplicity.

Although gossip-based coordination is based on local interactions and provides some context information to the agents, implementations very often assume that the communication range of an agent covers all other agents and all nodes can communicate with each other directly.

3.3.3 Tag-based Coordination

Another approach for decentralized coordination, inspired by social human behavior as well, is tag-based coordination [Hol93]. Tags are labels, markings, or social cues, which are attached to individual agents and are observable (and maybe addable or adjustable) by other agents. Here, coordination emerges because agents can discriminate based on the observed tags of others. An example is *group formation* [Hal06]. Groups of agents can be formed around similar tags, presumed that agents interact preferentially with other agents sharing the same tag. By using and adjusting the right tags, a desired group cooperation can be achieved. Another example is *self-protection* in terms of trust and reputation [Ser04], sometimes also known as trust-based coordination, which can be implemented by tags. After each interaction, an agent updates its local trust value on the interaction partner and may additionally also request or receive recommendations about other agents. Trust and reputation can be enforced by excluding agents from interactions that are tagged as badly behaving agents, presumed that tags on an agent can only be added and adjusted by other agents. Thus, over time, trust evolves as a result of updating and recommending and allows to adapt the behavior of agent in a self-organizing manner. Trust calculation algorithms, e. g. EigenTrust [KSGM03], then allow to calculate a global emergent reputation from locally maintained trust values. Tag-based interaction can be supported by an active agent environment [PSH07].

3.3.4 Token-based Coordination

In general, tokens are objects that may encapsulate anything that needs to be shared by a group of autonomous elements, including information, tasks, and resources [XSY+05]. For each token type there exists a limited number of instances and the element holding a token has exclusive control over whatever is represented by that token. Hence, tokens provide a type of access control that can be used for e. g. task and *resource allocation* as well as *self-protection*. An element holding a resource token has exclusive access to the resource represented by that token and passes the token to transfer access to that resource. If a token represents a certain role in the group, token-based coordination can be used to achieve *role-based organizations*, as by fixing the number of tokens the number of entities in a certain role can be limited and an adaptive organizational structure can be enforced.

Although the coordination process itself is decentralized and in certain situations minimizes communication effort compared to other decentralized coordination approaches, e. g. market-based coordination (cf. [XSSL06]), very often token-based coordination mechanisms are still based on direct interactions and require that all participating agents share a top level common goal (see e. g. [XSY+05]), which contradicts the principles of emergence.

3.3.5 Immunity-based Coordination

An approach for decentralized coordination inspired by the human immune system is immunity-based coordination, which is based on the theory of immunological computation [DN08]. The human or in general biological immune system is constituted by a collection of specialized and inter-related organs, cells, and molecules, which are distributed throughout most parts of an organism. These constituents are in charge of distinguishing "self" from "non-self", i. e. normal and anomaly situations. The human immune system is inherently distributed and fault-tolerant, and exhibits a complex behavior while interacting with all of its constituents. The immune system works on two levels: Innate immunity on the one hand is inborn and unchanging. It provides resistance to a variety of antigens during their first exposure to a human body. On the other hand, acquired immunity develops during the lifetime of a human. This immunity is specific to antigens and is activated during the first exposure to antigens.

Similarly, in artificial immunity-based coordination systems, each immune element (agent, robot, ...) has its own capabilities that determine its basic behavior as well as specific behavior for a particular response to antigens (resources, tasks, ...). The elements identify antigens during random exploration of the dynamic environment using an affinity measure to verify their feasibility to tackle an antigen (*self-protection*). The elements communicate within a limited range directly with each other to exchange local information, to send cooperation signals (such as "request for" and "respond to" help strategies), and to transfer capabilities, in order to achieve common tasks.

3.3.6 Pheromone-based Coordination

A decentralized coordination approach inspired by nature, in more detail by ants, is pheromone-based coordination [Brü00] (see also Subsection 4.5.2, later). This approach utilizes the principle of quantitative stigmergy (see Subsection 2.3.2) to design *spatial shapes* on the macroscopic level. The model is inspired by the mechanism ants (among others) use to find food. A pheromone is a chemical substance an ant can drop in the environment. The pheromone then propagates through the environment, as well as evaporates over time. Following ants for example use aggregated pheromones to build a path from the nest to a food source. In the computational world, typically, digital (aka synthetic) pheromones are used for constructing paths, which autonomous agents can follow. Therefore, spatial properties such as routing and the optimization of those routes can be achieved. Pheromone-based coordination mechanisms are inherently adaptive because old and not reinforced pheromones (i. e. old information) gradually disappear or evaporate, which constitutes an ongoing truth-maintenance mechanism. Appropriate infrastructures such as PI [Brü00], Anthill [BMM02], or SwarmLinda [MT03, TM04], fully support *indirect interactions* between the agents, account for context-awareness and meaningful interactions, and provide a spatial and temporal uncoupling of the agents.

3.3.7 Field-based Coordination

Another stigmergy-based coordination approach inspired by nature, however by physics, in more detail by magnetic fields, is field-based coordination (or computational field respectively co-field coordination) [MZ04, MZ06]. The basic idea of this decentralized coordination approach is that autonomous elements spread out computational fields through the environment. The field forms a gradient map, which conveys useful context information for the elements' coordination tasks. The coordination policy is realized by letting autonomous agents move following the waveform of these fields, e. g. uphill or downhill. Environmental dynamics and movement of entities induce changes in the fields' surface, composing a feedback cycle that influences how entities move. This feedback cycle lets the system adaptively self-organize. This coordination mechanism is promising to guide spatial movement of agents and as such construct *spatial shapes* and enforce *spatial distributions*. However, the coordination approach apparently can be used for *information dissemination* as well. Infrastructures such as TOTA [MZL03, MZ09] similarly fulfill the environment requirements for decentralized coordination by *indirect interaction* in self-organizing emergent MAS systems.

3.3.8 Bottom Line

Apparently, almost all decentralized coordination approaches got their inspiration from existing paradigms in fields such as economics, social human behavior, biology, or physics. This underpins the approach taken in this thesis, to observe an existing paradigm that achieves some required criteria in nature and to try to reverse-engineer its strategy. An exception of this approach is made by token-based coordination, which is not based on a comparable paradigm and hence very often requires that all participating agents share a top level common goal, contradicting the principles of emergence. However, some token-based coordination approaches exist that do not have this assumption, e. g. in P2P routing mechanisms.

3.4 Engineering Methodologies

Whereas appropriate decentralized coordination models and mechanisms respectively objective and subjective coordination are key issues for the design of self-organizing emergent MASs, the design phase itself is always embedded in an engineering methodology. Consequently, any methodology for the engineering of self-organizing emergent MASs should necessarily exploit both objective and subjective coordination during the design phase (cf. [OO03]). Apart from that, any methodology for the engineering of self-organizing emergent MASs should necessarily also exploit both the (micro-scale) behavior on the local level, i. e. the agents along with their internal rules, their interactions, etc., and the (macro-scale) emergent behavior on the global level [ZO04].

Although the engineering of conventional MASs has put various methodologies

forth, e. g. MaSe [DWS01], SODA [Omi01], PASSI [CCG+02], MESSAGE [BC02], GAIA [ZJW03], ROADMAP [JPS02], TROPOS [GKMP04], Prometheus [PW04], INGENIAS [PGSFF05], or ADEM [Whi07], they are insufficient for engineering self-organizing emergent MASs. The reason is that these methodologies mainly focus on the engineering of the behavior on the local level without explicitly engineering the required emergent behavior on the global level.

Bernon et al. [BGPP02] proposed one of the few exceptions, namely ADELFE, which is an engineering approach explicitly exploiting emergence among a set of cooperative agents, even if it is restricted to a specific class of MASs, in this case the AMAS (Adaptive Multi-Agent Systems) theory [CGGG03]. ADELFE uses Unified Modeling Language (UML) and Agent UML (AUML) [BMO01] notations and is based on the Unified Process (UP) [JBR99]. The UP is an iteration-based, incremental software development process framework that may be customized for specific organizations or projects. To be specific to the AMAS theory, some steps were added to the classical UP workflows by ADELFE. Similar to MESSAGE, PASSI, and TROPOS, ADELFE covers the entire software engineering process, however, contrary to e. g. GAIA or TROPOS, is not a general methodology but rather concerns applications in which self-organization makes the solution emerge from the interactions of its parts. The three essential workflows of ADELFE are the *requirements workflow*, the *analysis workflow*, and the *design workflow*.

The key part of the *requirements workflow* is the definition of an environment model, which consists of determining the actors, defining the context, and characterizing the environment. The *analysis workflow* mainly identifies agents and studies interactions between them. The *design workflow* studies the interactions between the agents and the environment and defines the agents' behavior (skills, aptitudes, etc.). The design workflow has been subsequently extended by a simulation stage, to modify and improve the behavior of agents during the design [BGP06].

De Wolf and Holvoet [DH05] similarly address the shortcomings of agent-oriented software engineering methodologies and focus explicitly on the engineering of macroscopic properties of self-organizing emergent MASs. They propose a way – considered as a starting point to integrate future work – to define a full life cycle methodology, which is also based on iterations defined by the UP. In contrast to ADELFE, this methodology is not limited to the AMAS theory, but supports the engineering of general MASs. Basically, each iteration includes the four process disciplines *requirements analysis*, *design*, *implementation*, as well as *verification and testing*. The focus of each discipline is on how to address the desired macroscopic properties. With each iteration the self-organizing emergent MAS becomes successively refined, with cyclic feedback from verification and testing to design.

The *requirements analysis* emphasizes an investigation of the problem with functional and non-functional requirements and pays attention to issues that typically lead to self-organizing emergent MASs with macroscopic behavior, such as ongoing and adaptively maintained requirements – the self-* properties. The *design* emphasizes a conceptual solution that fulfills the identified requirements and is split into two phases: In the *architectural design phase* (early iterations) the focus is more

on the coarse-grained software architecture [SG96, CBB+02], exploiting knowledge and experience (design principles, reference architectures, etc.) from existing best practice in engineering self-organizing emergent MASs. In the *detailed design phase* (later iterations) the architecture converges more and more to a fixed structure and the more fine-grained design issues are resolved, i. e. mainly the microscopic behavior of agents. The *implementation* realizes the previously specified (hardware and software) design in code. This requires no special customizations for macroscopic issues because the implementation is completely microscopic. In early iterations of *verification and testing* the focus is more on the coarse-grained decisions with respect to the architecture of the solution (analysis of non-functional and macroscopic performance, comparison of coordination mechanisms, parameter tuning, etc.). In later iterations the focus is more on the details of the solution, i. e. setting and changing parameters to adaptively react to changing situations.

Gershenson [Ger07] proposes a very general methodology for designing and controlling self-organizing emergent systems developed to solve complex problems. In contrast to the above approaches, the methodology aims to cover not only MASs but any kind of self-organizing emergent system. The methodology includes the five phases *representation*, *modeling*, *simulation*, *application*, and *evaluation*, which are arranged similar to a waterfall model with feedback. The methodology is not to be seen as a recipe that provides ready-made solutions, but rather as a guideline to direct the search for them.

The goal of the *representation phase* is to develop a (possibly tentative) specification of the components of the system. The designer should try to divide a system into elements by identifying semi-independent modules, with internal goals and dynamics, and with few interactions with their environment. In the *modeling phase*, an internal, distributed, and adaptive control mechanism is to be specified that will ensure the proper interaction between elements of a system producing the desired performance. The aim of the *simulation phase* is to build computer simulation(s) that implement the developed model(s) and test different scenarios. Because the precise behaviors of a complex system cannot be easily deduced from its model(s), i. e. they are not reducible, simulation is a key phase. In other words, a model needs to "run" before it can be understood. Based on the simulation results and insights, the modeling and representation can be improved. The role of the *application phase* is basically to use the developed and tested model(s) in a real system (which will be not difficult, if the real system is a software system, because the software would have been developed in the simulation phase already). In the *evaluation phase*, the performance of the system should be measured and compared with the requirements, while efforts should be continued to try to improve the system.

Serugendo et al. [SFRG08] present a framework for the engineering of dependable self-adaptive and self-organizing systems. The framework assumes that the later system will comprise autonomous components such as agents, however wrapped by services in a Service Oriented Architecture [SH05b]. For design time, the framework uses the three phases *analysis*, *design*, and *implementation*. During the *analysis phase*, desired properties of the overall system are identified, i. e.

functional aspects and non functional aspects such as self-* requirements and properties (e. g. self-protection or self-healing) as well as QoS. Driven by the identified properties, during the *design phase* the engineer first selects adequate architectural patterns and identifies the coordination mechanisms that the system will adhere to. Second, the chosen architectures and patterns are refined for the specific application and the individual components (agents, services, etc.) are identified and defined. During the *implementation phase*, the run-time infrastructure is developed, together with the individual components and some other data (executable policies and metadata sensing and monitoring capabilities).

Gardelli et al. [GVCO08] provide a design approach that can be plugged between the analysis and the design phase of existing agent-oriented methodologies, hence realizing an *early-design phase*. The approach is based on the A&A metamodel [ORV08], which describes a MAS in terms of agents and artifacts. For the approach they assume that requirements have been collected and the analysis has been performed. The early-design phase comprises the three phases *modeling*, *simulation*, and *tuning*.

In the *modeling phase* an abstract specification of the system is developed, preferentially by formal languages. In the *simulation phase* these specifications are used to qualitatively and quantitatively investigate the dynamics of the system, i. e. the global system behavior is tested in different environmental conditions that are representative of expected or actual scenarios. In the *tuning phase*, the agents' behavior and working parameters are successively tuned until the desired dynamics are observed. This may require backtracking the modeling choices and evaluating other modifications or approaches until the performance of the system is satisfying. Critical systems may require an additional in-depth formal analysis before proceeding to the actual design phase.

3.5 Conclusion

In this chapter we have provided the background for and state of the art in designing self-organizing emergent systems. Although methodologies for the engineering of these systems are scarce yet (see last section), a few major issues for the design phase become quite evident.

Because a straight forward engineering of self-organizing emergent (multi-agent) systems is not possible due to the fact that their precise macroscopic behavior cannot be easily deduced from their microscopic models (cf. *Problem 1*), *simulation* has become a key phase in the engineering of self-organizing emergent systems, either before (see [GVCO08]), during (see [BGP06]), or after (see [DH05, Ger07]) the design phase. Edmonds [Edm05] even argues that simulations and experiments are strictly necessary in the design of self-organizing emergent systems, because their performance (considered as macroscopic property) cannot be evaluated by purely formal methods [EB04]. Similarly, De Wolf [De 07] notes that although the macroscopic behavior can still be specified formally, it is practically infeasible to formally

verify or proof the correctness of the macroscopic behavior. Thus, if the simulation results are not satisfying, the design of the systems can be adapted, either by backtracking (see [Ger07, GVCO08]) or cyclic feedback (see [DH05]), until the results are satisfying. Emergence Engineering, which tries to utilize emergent phenomena even within the engineering process, e. g. [ZW07], also depends fundamentally on a simulation phase. As a direct consequence, appropriate simulation tools, such as the one we will present in Section 8.1, are required. They can be integrated into engineering methodologies for self-organizing emergent MASs and support engineers in identifying and selecting the most suitable and efficient decentralized coordination approaches according to the required functional and non-functional criteria for a system (cf. *Challenge 3*).

Table 3.1 compares the decentralized coordination approaches listed in Section 3.3 based on such criteria, as they are described at the beginning of that section. Although all of these coordination approaches enable the design of self-organizing emergent solutions for various application scenarios, the source of inspiration of most of these approaches is "exhausted", i. e. it does not serve for the inspiration and specification of new coordination mechanisms based on these approaches. For instance, whereas the principles of gossiping are captured in the gossip-based coordination approach, the design of new coordination mechanisms based on this approach can not be inspired from existing gossiping mechanisms in the human society, because there are no more. Instead, they have to be designed from scratch in a complex and time-consuming process to overcome these lacks, which also requires some kind of expert knowledge in gossiping from engineers (cf. *Challenge 2*). An exception is made by pheromone-based coordination, as the identification of new biological paradigms using pheromones may be used for the specification of new decentralized coordination mechanisms. A further, smaller exception is made by market-based coordination, as new types of auctions could be adopted as decentralized coordination mechanisms. However, waiting for new auction types to emerge in economics is not appropriate for most problems.

Due to the comparison made in Table 3.1, it becomes apparent that an required solution to a problem in hand may exceed the provided functionality of a single decentralized coordination approach (cf. *Challenge 1*). For instance, de Wolf and Holvoet [DH07b] thus had to combine field-based coordination with market-based coordination in order to achieve routing and dispatching of packets in a packet delivery service application. Weyns et al. [WBH06] developed a field-based coordination approach for adaptive task assignment, but had to combine this model with a hand-made model for a collision avoidance mechanism for the agents.

Furthermore, very often the expressiveness of most coordination approaches is too constraint and not sufficient to design efficient solutions (cf. *Challenge 2*), which very often requires time-consuming hand-made extensions or variations. For instance, pheromone-based coordination does not allow for different functions, dynamics, and semantics of digital pheromones or even different types of agents. Thus, to improve the performance of a pheromone-based coordination solution, Brückner and Parunak [BP00] had to use multiple types of pheromones with differing dynamic character-

	Market-based	Gossip-based	Tag-based	Token-based	Immunity-based	Pheromone-based	Field-based	Infochemical-based
Resource allocation	+			+				+
Group formation			+					+
Organizations				+				
Self-protection			+	+	+			+
Information dissemination		+					+	+
Spatial shapes		+				+	+	+
Spatial distribution	+						+	+
Indirect interaction	(+)					+	+	+
Source of inspiration	(+)					+		+

Table 3.1: Comparison of decentralized coordination approaches

istics, e. g. different propagation radii, rates, and thresholds. Later, Parunak et al. [PBS04] did not only experiment with pheromones having different dynamics but also with pheromones having different semantics to increase system performance. Similarly, Panait and Luke [PL04], Sauter et al. [SMPB05], as well as Brückner and Parunak [BP05] had to use pheromones with different semantics to design a solution of acceptable efficiency. Di Caro and Dorigo [DD98] in contrast had to use different types of ant agents to design a working solution. Furthermore, they therefore had to use a more general interpretation of a pheromone, as they considered routing tables in a network as pheromones. Similarly, Valckenaers et al. [VKvB+01] designed a solution, in which they updated loading profile data on workstations of a factory in a pheromone style.

Due to these inadequacies, in the next chapter we present the infochemical-based coordination approach that helps to overcome these drawbacks (see Table 3.1). The IBC approach is based on the biological principles of coordination by infochemicals, which are chemical stimuli used to mediate the indirect interactions between organisms in nature. Because infochemical-based coordination is the most universally employed communication and coordination model between homogeneous and heterogeneous organisms in biology, it provides a plethora of inspiring examples for the identification of and inspiration for new decentralized coordination mechanisms. Furthermore, the terminology and expressiveness of infochemicals comprises pheromones as well as further types of chemical stimuli with different functions, dynamics, and semantics, which allows for the combination of different types of infochemicals within one coordination mechanism as well as a combination of quantita-

tive and qualitative stigmergy. This in turn allows for the design of better adaptable and efficient solutions compared to existing decentralized coordination approaches and only requires a single coordination approach to achieve a bigger functionality.

Chapter 4

Infochemical-based Coordination

The huge amount of different design aspects, which we have described in the last chapter, illustrates, why the design process of efficient self-organizing emergent MASs today is too complex, time-consuming, and costly for their widespread application in various areas (cf. *Problem 1*). Thus, in this chapter we present a couple of artifacts in order to simplify the design process of these systems and to engineer effective as well as efficient solutions for individual problems (cf. *Objective 1*). As pointed out in the last chapter, the key issue that has to be tackled here is the appropriate and efficient decentralized coordination of the interactions between the system elements.

In general, the coordination of interactions between individuals is an omnipresent problem that is not only pertinent to computational ones such as agents but also to individuals of other disciplines, such as living organisms in nature for instance, in more detail animals or plants. There, the purpose of coordination, by contrast, is rather the selection of food, the selection of mates, competition, or the avoidance of predators. Living organisms thereby effectively rely on different, well-elaborated mechanisms of communication and coordination based either on radiational (light perception or visual), mechanical (tactile or auditory), or chemical (gustatory or olfactory) stimuli. These mechanisms are purely decentralized and consequently enable a high degree of scalability, robustness, flexibility w. r. t. changes in the population, and adaptivity w. r. t. changes in the environment. Thus, the understanding of the principles behind such mechanisms and, if applicable, their computational adaptation is of high importance to engineer effective self-organizing emergent systems.

In this chapter, we investigate in more detail the general principles behind the communication and coordination by means of chemical stimuli, more precisely *infochemicals* [DS88]. Infochemical-based coordination is the most universally employed communication mechanism in nature (cf. [Lew84]) providing a plethora of inspiring examples for decentralized coordination. Basically, infochemicals in the natural context mediate interactions between living organisms. They are divided into two broad categories: *pheromones* that mediate intraspecific interactions, i. e. between organisms of the same species, and *allelochemicals* that mediate interspecific interactions, i. e. between organisms of different species. Whereas the principles behind intraspecific coordination by means of pheromones have been already extensively studied and used in the computational world (see e. g. [Brü00, VHG+07] as well as Paragraph 3.3.6), the principles behind interspecific coordination by means of allelochemicals,

e. g. between plants and insects, have been fairly neglected, yet. However, we will see that the combination of different types of infochemicals within one single coordination mechanism allows for the design of better adaptable and efficient solutions compared to existing coordination approaches.

Section 4.1 investigates the general principles behind infochemical-based coordination in nature. Section 4.2 then presents the formal adaptation of these biological principles by a coordination model for self-organizing emergent MASs. Based on this adaptation, Section 4.3 presents a structured design pattern that supports the systematical engineering of self-organizing emergent MASs, whereas Section 4.4 presents design guidelines how to use and instantiate the infochemical-based coordination model according to specific needs. Section 4.5 then exemplary specifies pheromone-based coordination as a specific instantiation of the more general infochemical-based coordination. Finally, Section 4.6 mentions some related work to infochemical-based coordination, before Section 4.7 concludes this chapter.

4.1 Principles

In general, three factors are necessary for the occurrence of communication between two organisms: one of the organisms must emit a message, a medium must be available through which the message is transmitted, and the other organism must respond in some way when exposed to the message, which is then called a stimulus [Sho73]. Communication and coordination by chemical stimuli, i. e. olfactory or gustatory stimuli, is a universal feature of life that occurs at all levels of biological organization, including the movement of cells or bacteria (called chemotaxis), the regulation of organs within an individual's body by hormones, as well as social behavior and ecological interactions among individuals by so-called semiochemicals or infochemicals. In the animal world, olfactory communication is probably one of the oldest and, in some cases, the most efficient means of communication [RL68]. In insects, it is even the most universally employed mode of communication [Lew84]. For example, odor can reveal much information about an individual, including its sex, diet, social status, individual and group identity, reproductive condition, age, health, fear, and other emotional states [Wya03].

The ecological understanding of the origin, function, and significance of the natural chemicals that mediate interactions within and between organisms are main subjects in chemical ecology. From the beginning of chemoecological research in the 1950s until now, several attempts have been undertaken to create a unique terminology for these chemicals. Reviews of the terminology can be found in [Duf76] and [Nor81], for instance. Although the chemicals are classified according to their function or effect in specific interactions [WF71], their functions are not mutually exclusive [BEW70]. Thus, a given chemical can have several biological functions within a complex network of interactions, which makes its classification rapidly complicated. In addition, the permanent progress in the identification of new chemical compounds in ecological interactions exacerbates a clear classification.

4.1.1 Terminology

As already mentioned, chemically mediated interactions between organisms in biology can be classified as intraspecific, i. e. between organisms of the same species, or interspecific, i. e. between organisms of different species. In biology, a species can be defined as a group of actually or potentially interbreeding populations that are reproductively isolated from other such groups. As such, a species forms the basic level of all main taxonomic ranks[1].

Regarding the chemicals mediating these interactions, in 1971 Law and Regnier proposed the term *semiochemicals* (Greek *semeion*, a mark or signal) and defined them intuitively as *"chemical signals for transmitting information between individuals"* [LR71]. However, the word 'signal' presupposed that the chemical is purposefully sent to a receiver, but as the receiving organism commonly acts as an eavesdropper only, Nordlund and Lewis eliminated 'signal' from the definition in 1976 and redefined a semiochemical as *"a chemical involved in the chemical interaction between organisms"* [NL76]. They subdivided semiochemicals into two major categories: *pheromones*, for intraspecific interactions, and *allelochemicals*[2], for interspecific interactions. Allelochemicals can be further subdivided into three subcategories: *allomones*, *kairomones*, and *synomones*. A distinction between these subcategories is made, whether the emitting organism (allomones), the receiving organism (kairomones), or both organisms (synomones) benefit in the interactions. Originally, Nordlund and Lewis distinguished a fourth subcategory termed *apneumones*, which is defined as *"a substance emitted by a nonliving material that evokes a behavioral or physiological reaction adaptively favorable to a receiving organism, but detrimental to an organism, of another species, which may be found in or on the nonliving material."* [NL76]. However, this subcategory has not been used since it was proposed, possibly because it is difficult to distinguish these chemicals from allelochemicals that are produced by micro-organisms on the nonliving material.

The terminology by Nordlund and Lewis uses two criteria for the classification of a chemical: (1) whether the emitter (the organism that is the origin of the chemical) and the receiver are conspecific, i. e. from the same species, or not, and (2) whether the emitter, the receiver, or both benefit in the interactions mediated by the chemical. In this widely accepted terminology the origin of the chemical is of central importance. However, as the investigation of the exact origin is difficult (often micro-organisms living on an organism act as producer of a chemical, see e. g. apneumones), the proposed distinction of semiochemicals often leads to ambiguities and heterogeneous definitions. Furthermore, the cost-benefit analysis is limited to allelochemicals only and no cost-benefit classification existed for pheromones on the individual level. Thus, in 1988 Dicke and Sabelis [DS88] proposed to eliminate the origin criterion and rather base the classification of the chemicals, therein called *infochemicals*, on the cost-benefit criterion only.

[1] There exist eight hierarchically-organized, main taxonomic ranks: species, genus, family, order, class, division, kingdom, and domain (cf. [Int99]).

[2] The term *allelochemical* has generally replaced the original term *allelochemic* used in [NL76].

An infochemical is defined as *"a chemical that, in the natural context, conveys information in an interaction between two individuals, evoking in the receiver a behavioral or physiological response that is adaptive to either one of the interactants or to both"* [DS88]. Infochemicals are subdivided into the two categories pheromones and allelochemicals as well (see Figure 4.1). Whereas the latter category is subdivided into allomones, kairomones, and synomones too, the former category is analogously subdivided into (+,-), (-,+), and (+,+) pheromones, in order to take cost and benefits on the individual level of the emitter and the receiver into account. In the next paragraphs we investigate each of these categories in more detail.

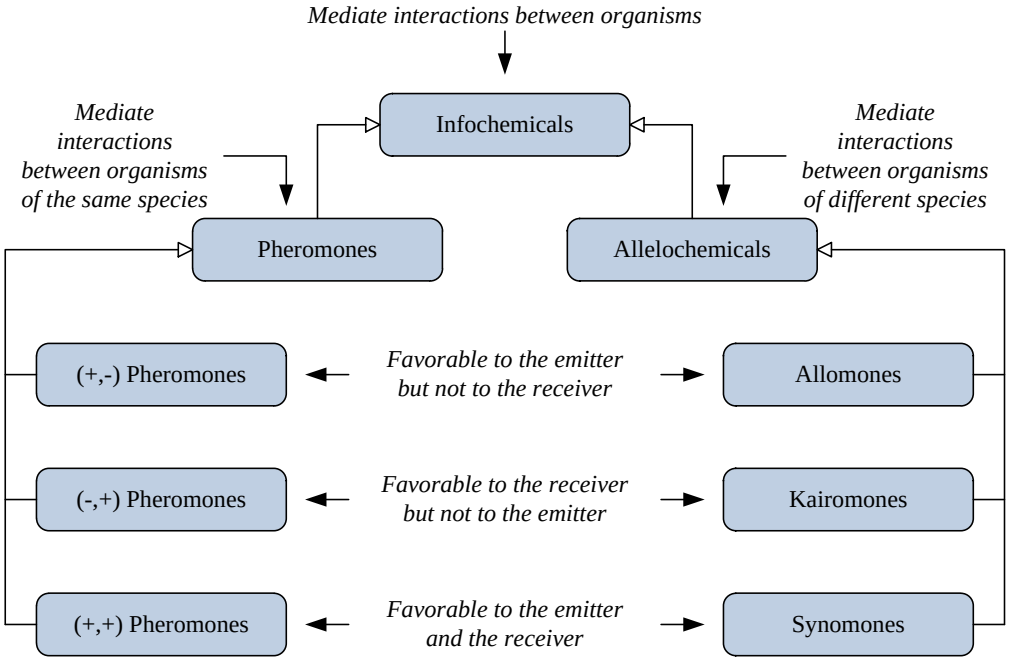


Figure 4.1: Classification of infochemicals

Pheromones

Originally, pheromones (Greek *pherein*, to transfer, and *hormōn*, to excite) were defined in 1959 by Karlson and Lüscher as *"substances which are secreted to the outside by an individual and received by a second individual of the same species, in which they release a specific reaction, for example, a definite behavior or a developmental process"* [KL59]. At this time, the words 'to the outside' were the most important part, as they distinguish pheromones from *hormones*, which are chemicals produced by tissues or endocrine glands that cause specific reactions within the producing organism. In 1976, the definition then was modified by Nordlund and Lewis as *"substances that are secreted by an animal or plant to the outside that cause a specific reaction in a receiving individual of the same species"* [NL76].

Finally, in 1988 Dicke and Sabelis redefined a pheromone as *"an infochemical that mediates an interaction between organisms of the same species whereby the benefit is to the origin-related organism ([+,-] pheromone), to the receiver ([-,+] pheromone) or to both ([+,+] pheromone)"* [DS88]. As the term pheromone had been widely accepted already, they qualified a pheromone with '(+,-)', '(-,+)', or '(+,+)' where respectively the origin-related organism, the receivers, or both benefit.

Beside ant foraging, an example of pheromone-mediated interaction is demonstrated by a special species of a cannibalistic snail: This snail produces a pheromone that may be used to distinguish different sizes of conspecifics, i. e. members of the same species. A snail moves toward conspecifics with a size smaller than its own and away from conspecifics with a size larger than its own. For a large receiver snail the infochemical involved is a (-,+) pheromone, as is the case for an interaction where the large snail is the emitter and a small one the receiver. Another example of pheromone-mediated interaction is the nest building in termite colonies.

Allelochemicals

The term allelochemic(al) (Greek *allelon*, one another, and *hormōn*, to excite) was originally proposed in 1970 by Whittaker [Whi70b, Whi70a] and defined as *"a chemical significant to organisms of a species different from its source, for reasons other than food as such."* Whereas Nordlund and Lewis [NL76] extended and clarified this definition by the introduction of the four subcategories mentioned above, Dicke and Sabelis modified the definition of an allelochemical as *"an infochemical that mediates an interaction between two individuals that belong to different species"* [DS88] and reduced the subcategories by one to the set allomones, kairomones, and synomones.

Allomones

The term allomone (Greek *allos*, another, and *hormōn*, to excite) was first proposed by Brown [Bro68] in 1968 and later defined by Nordlund and Lewis as *"a chemical substance, produced or acquired by an organism, which, when it contacts an individual of another species in the natural context, evokes in the receiver a behavioral or physiological reaction adaptively favorable to the emitter"* [NL76]. Dicke and Sabelis then redefined an allomone as *"an allelochemical that is pertinent to the biology of an organism (organism 1) and that, when it contacts an individual of another species (organism 2), evokes in the receiver a behavioral and/or physiological response that is adaptively favorable to organism 1 but not to organism 2"* [DS88].

An example of allomone-mediated interaction is given by the black walnut tree. It secretes juglone, a chemical that harms or kills some species of neighboring plants, from its roots. By removing competition and allowing the tree access to greater scarce resources, this interaction increases the fitness of the tree and harms the fitness of the neighboring plants. Other examples are plants that emit chemicals (toxins) to deter herbivores as well as skunks that emit chemicals to keep putative predators away.

Kairomones

The term kairomone (Greek *kairos*, opportunistic) was first proposed in 1970 by Brown et al. to describe *"a transspecific chemical messenger, the adaptive benefit of which falls on the recipient rather than on the emitter"* [BEW70]. This definition was modified by Nordlund and Lewis as *"a substance, produced, acquired by, or released as a result of the activities of an organism, which, when it contacts an individual of another species in the natural context, evokes in the receiver a behavioral or physiological reaction adaptively favorable to the receiver but not to the emitter"* [NL76]. Dicke and Sabelis finally redefined a kairomone as *"an allelochemical that is pertinent to the biology of an organism (organism 1) and that, when it contacts an individual of another species (organism 2), evokes in the receiver a behavioral and/or physiological response that is adaptively favorable to organism 2 but not to organism 1"* [DS88].

An example of kairomone-mediated interaction is given by bark beetles. These beetles have evolved a pheromone that elicits behavior resulting in aggregation of the population on a new host, e. g. living, recently killed, or fallen trees. However, the pheromones, in this role then referred to as kairomones, in the same way also attract predators and hence benefit natural enemies of the bark beetles. In the same way chemical cues released from mammals affect the behavior of mosquitoes, attracting them from a distance.

Synomones

In 1971, Whittaker and Feeny [WF71] classified allelochemicals that benefit the receiver as well as the emitter as both allomones and kairomones. As this led to ambiguities, Nordlund and Lewis introduced the term synomones (Greek *syn*, with or jointly) and defined a synomone as *"a chemical substance produced or acquired by an organism, which, when it contacts an individual of another species in the natural context, evokes in the receiver a behavioral or physiological response adaptively favorable to both the emitter and the receiver"* [NL76]. Dicke and Sabelis then redefined a synomone as *"an allelochemical that is pertinent to the biology of an organism (organism 1) and that, when it contacts an individual of another species (organism 2), evokes in the receiver a behavioral and/or physiological response that is adaptively favorable to both organism 1 and 2."* [DS88].

At the present time the number of examples of synomone-mediated interactions is still low. An example however is given in the pollination process of plants. Plants emit floral scents that attract insects and other pollinators to its location. While plants benefit in this interaction by the receipt of pollen grains from other plants, the pollinators benefit in this interaction by the collection of nectar or oils as a reward for their visit (see also later Subsection 7.2.1).

4.1.2 Functions and Effects

Apart from their benefits to origin-related or receiving organisms, infochemicals are also often divided by their function in the interactions between organisms. With regard to the category of pheromones, these infochemicals in most cases function by influencing other members of the same species, not the individual that produced them. Because the functions of infochemicals in general are not mutually exclusive but depend on the ecological context, various, partly overlapping, functional classes of pheromones can be found in literature (see e. g. [Ago92, JM93, RL68, Sho76, Wya03]). A non-exhaustive list of functional pheromone classes includes:

- **Sex pheromones:** The most thoroughly documented cases of long-range chemical communication are those of sex substances used in signaling a mating partner. For example, female moths release chemical stimuli into the air to signal their availability, and thereby attract males over long distances.

- **Alarm pheromones:** Used primarily by social animals to warn other members in the colony of impending danger or any threatening situation. The behavior of most animals upon reception of an alarm signal is basically the same. They initially orient osmotactically to the source at low pheromone concentration and at high concentration go into frenzied activity, occasionally attacking the pheromone source.

- **Aggregation pheromones:** Used for causing other members of the same species to aggregate in a particular area, e. g. a food source or a suitable habitat. The aggregations may be dense, e. g. thousands of bark beetles arriving at a designated host tree, or not dense, e. g. a single male arriving at the vicinity of a female that is signaling her readiness to mate (in terms of sex pheromones). The behavioral response of an insect stimulated by the aggregation pheromone is movement toward the pheromone source and/or cessation of locomotion, at least temporarily, once the insect has arrived at the source.

- **Dispersal or spacing pheromones:** Used to increase the spacing between conspecifics, and thus reduce intraspecific competition. They may be used to prevent overcrowding of resources such as food, mates, egg-laying sites, and refugia.

- **Home range pheromones:** Used to mark an area within which an organism normally confines its activities. When this area or territory is defended against other organisms, the pheromones are called *territorial marking pheromones*.

- **Trail pheromones:** Used by many insects, especially by ants, for orientation to food sources or new nest sites. Wilson [Wil71] described the odor trail system as the most elaborate of all known forms of communication in social insects.

- **Surface pheromones:** Used in social insect colonies and allow recognition of nest mates, kin, or even members of different castes. This broad class also includes recognition pheromones, releasers of grooming behavior and secretions that stimulate food exchange.

The perception of a pheromone may result in an immediate behavioral response (releaser pheromones) or a complex set of physiological responses that are simply set in motion by the initial perception (primer pheromones) [WB63]. Primer pheromones trigger physiological changes in the recipient, which then equip the organism with a new behavioral repertory. Very often sex pheromones function as primer pheromones for instance. The effect of primer pheromones is generally long-term, without an obvious immediate response. Releaser pheromones in contrast cause an immediate and reversible change in behavior mediated directly by the central nervous system. All above mentioned classes may function as releaser pheromones.

With regard to the category of allelochemicals, a differentiation between releaser and primer allelochemicals is also possible, although not all allelochemicals are further subdivided. Ruther et al. [RMS02] have proposed a classification of kairomones according to the function for the benefiting organism, i. e. the receiver. This classification comprises four main classes, whereas the first three classes attract the receiving organism and the fourth class repels it:

- **Foraging kairomones:** Used by the benefiting organism in the context of food location, e. g. volatiles used by herbivores, parasitoids, parasites, or fungivores to locate hosts or host plants.
- **Sexual kairomones:** Used by the benefiting organism for sexual purposes, e. g. plant volatiles used to locate feeding mates or enhancing response toward pheromones.
- **Aggregation kairomones:** Used by both sexes of the benefiting organism to form aggregations, e. g. for optimal exploitation of food resources, mate finding, or as defense reaction.
- **Enemy-avoidance kairomones:** Used by the benefiting organism to recognize the presence of natural enemies, e. g. predator-borne volatiles inducing escape reaction in potential prey.

Classified according to the effect on the benefiting organism, all classes can be regarded as releaser kairomones. However, sexual as well as enemy-avoidance kairomones may also be considered as primer kairomones [RMS02]. Sexual kairomones may induce physiological reactions in the context of sexual behavior, e. g. plant volatiles inducing pheromone production and release. Enemy-avoidance kairomones may induce physiological reactions that reduce the negative impact of a natural enemy, e. g. predator-borne chemicals causing the development of defensive morphological structures.

According to [RMS02], the classification of kairomones can be transferred on allomones as well. Very often the term allomone is used in the context of typical defense chemicals which hence are enemy-avoidance allomones. Scents emitted by predacious organisms to lure their prey, e. g. in the aggressive chemical mimicry shown by bola spiders mentioned above, may be classified as foraging allomones. Those volatiles emitted by orchids mimicking the sex pheromones of their pollinators may be interpreted as sexual allomones, since these plants do not reward their pollinators and the responder does not benefit from this interaction. A differentiation between primer and releaser allomones is principally possible, too.

In the case of synomones, the transfer of the kairomone classification is difficult. In many cases, a synomone will have different ecological functions for the emitter and for the receiver. Hence, the criterion for the classification can not be defined unambiguously [RMS02].

4.1.3 Communication

From a chemical point of view, infochemicals are chemical compounds, which can range from highly volatile to non-volatile. Most compounds are of relatively low molecular weight and many are derivatives of fatty acids or terpenes. Slight genetic, dietary, and environmental differences make it improbable that any two organisms produce the same blend of volatile organic compounds. This probably accounts for the fact that many animals are able to identify their young or members of their own group in large assemblages of other individuals [RL68].

4.1.3.1 Production and Release

The production of an infochemical by living organisms is regulated through hormones and signal transduction pathways. In contrast to hormones, which are produced in the endocrine glands, infochemicals are produced and discharged from exocrine glands. They are either secreted onto a surface area (e. g. for trial marking) or in most cases released into the surrounding air forming a cloud of vapor about the releasing organism [BW63]. In a few cases, infochemicals are also released into aqueous systems.

4.1.3.2 Transmission

In general, the distance through which an infochemical may transmit information is a function of the volatility of the compound, its stability in air, its rate of diffusion, olfactory efficiency of the receiver, and wind currents [RL68]. Bossert and Wilson [BW63] have derived a mathematical model, which predicts the diffusion behavior of a volatile chemical in still air on the basis of these parameters. The concentration of a chemical at various distances from a point of release may be calculated as a function of time by the following equation:

$$U(d,t) = \frac{W}{2D\pi d} \cdot erfc(\frac{d}{(4Dt)^{0,5}}) \tag{4.1}$$

where

- U is the concentration of the chemical measured in molecules/cm^3
- W is the emission rate of vapor from the source in molecules/sec
- D is the diffusion coefficient of the chemical in air in cm^2/sec
- d is the distance from the emission source in cm
- t is the time from the beginning of emission in seconds
- $erfc$ is the complementary error function

Diffusion coefficients for most airborne pheromones are between 10^{-1} and 10^{-2} cm^2/sec, while the same compounds have diffusion coefficients in the range of 10^{-5} cm^2/sec in water [LR71]. Thus, long-distance communication of a mile or more must be mediated by the use of stable compounds with high vapor pressures.

4.1.3.3 Perception

Infochemical perception is usually considered to be an olfactory process, although in some cases it may be gustatory, for example with infochemicals, which are transmitted in aqueous media or which are non-volatile [RL68]. The perception of an infochemical then triggers an immediate behavioral response (releaser effect) or a delayed physiological response (primer effect), as described in Subsection 4.1.2. However, a behavioral response requires the concentration of an infochemical to be high enough. This concentration, in molecules/cm^3, is called the behavioral threshold concentration K [BW63].

4.1.4 Advantages

As already mentioned at the beginning of this chapter, living organisms interact by relying on different mechanisms of communication, more specifically visual, tactile, auditory, or chemical stimuli. Although chemical stimuli on their journey to a receiver have to pass through a highly variable environment affected by wind, temperature, moisture, and physical obstructions such as plants, animals, and rocks, the widespread use of chemical stimuli to interact with conspecifics as well as heterospecifics, i. e. members of different species, is indicative of the many advantages of this way of information conveyance (cf. [Dot86]):

1. It is obvious that infochemicals can be used in situations where visual or auditory signals are absent or difficult to discern, e. g. at night, in dark burrows, or near loud sound sources.

2. Infochemicals can be easily distributed in both space and time, making them uniquely suited to provide information about 'territoriality' or space occupancy. Thus, the distribution, concentration, and qualitative aspects of e. g. scent marks presumably provide conspecifics with key information about the resident(s) of an area, such as their stamina, physical condition, physical size, reproductive state(s), mobility, energy level(s), motivational tendencies, group constituency, and group size. The temporal aspects of infochemicals allow for the sending of 'time-coded' messages, such as the period of time since a given area has been visited or occupied, or specific information about the reproductive state.

3. Relative to other sorts of sensory stimuli, infochemicals can remain in the environment for rather long periods of time without jeopardizing the immediate safety of the signaling individual. If an animal released a continuous noise or visual signal in a manner analogous to leaving a long-lasting infochemical, not only would an inordinate amount of energy be expended, but predators would have a field day (literally) in locating it. This long-lasting property of infochemicals allows a dominant male, for example, to make his odor more or less continuously present in the social environment.

4. The sender and receiver need not be in close proximity for the communication to take place. This permits a resident to communicate to an intruder or rival that a given space is occupied, or that he or she is reproductively active, even though they are outside the range of hearing or sight. Such communication minimizes the physical harm, expenditure of energy, and exposure to predation, which can result from direct physical encounters or inappropriately directed mating advances.

These advantages are also represented by the comparison of the characteristics of different modes of communication (see Table 4.1, according to [Lew84]).

Feature of mode	**Type of signal**			
	Visual	**Tactile**	**Auditory**	**Chemical**
Range	Medium	Short	Long	Long
Rate of change of signal	Fast	Fast	Fast	Slow
Ability to circumvent obstacles	Poor	Poor	Good	Good
Locatability	High	High	Medium	Variable
Energetic cost	Low	Low	High	Low

Table 4.1: Characteristics of different modes of communication

4.2 Coordination Model

In order to employ the aforementioned beneficial properties of infochemical-based coordination for the efficient decentralized coordination of agents in self-organizing emergent MASs, in this section we formally capture the general biological principles behind infochemical-based coordination in an decentralized coordination model. Such a coordination model represents a design artifact and will help to simplify the design of efficient self-organizing emergent MAS solutions. Please note that the purpose of biologically-inspired solutions in general is not to copy the exact natural functionality, but rather to utilize the principles of the biological paradigms (cf. e. g. [OZ05]), which is why we make some abstractions in the specification of the infochemical-based coordination model compared to the biological pendant.

As described in Section 3.2, a coordination model for MASs provides a formal framework in which the interaction of the agents can be expressed. It is composed of the coordination elements (coordinables), the coordination media, and the coordination laws. The coordination elements obviously are the agents themselves, whereas the coordination media provides abstraction for enabling and controlling the interactions among the agents. The coordination laws define how the events are handled by the coordination media when agents interact.

4.2.1 Coordination Elements

In the infochemical-based coordination model, an agent reflects a living organism, able to interact indirectly by means of (digital) infochemicals.

Definition 4.1 (Infochemical)
An infochemical ι is defined as a quintuplet

$$\iota = (\gamma, \gamma^{thresh}, \delta, \epsilon, \psi)$$

where

- $\gamma \in \Re$ is the concentration of ι
- $\gamma^{thresh} \in \Re$ is the threshold concentration of ι
- $\delta \in \Re$ is the diffusion coefficient of ι
- ϵ is the emitter of ι
- ψ are individual information encapsulated by ι

For the definition of a digital infochemical, we explicitly decided not to reuse the symbols used in the natural context (see Subsection 4.1.3.2), in order to prevent unnecessary confusion due to different measuring units or calculations. Thus, γ reflects the dynamically changing concentration of diffusing biological infochemicals,

referred to a U in the natural context (see Subsection 4.1.3.2). γ^{thresh} reflects the behavioral threshold concentration of living organisms (K) regarding specific infochemical types. Admittedly, according to the object-oriented paradigm, this attribute thus should be ideally modeled as an attribute of an agent itself. However, whereas in biology the diffusion of infochemicals proceeds up to the last molecule, this has no practical effect in the computational world, in particular not from an object-oriented perspective. Thus, if the current concentration of an infochemical falls below this threshold, i. e. $\gamma < \gamma^{thresh}$, it will not be propagated any further but removed immediately. δ reflects the chemical diffusion coefficient (D) and thus allows for a very fine-tuned propagation radius and evaporation time specific to each infochemical. ϵ reflects the biological fact of an infochemical to reveal information on the emitter. ψ reflects the biological role of an infochemical as a dynamic information carrier and depends on the used coordination mechanism respectively the application domain the solution has been designed for.

A set of infochemicals is denoted as $\mathcal{I} = \{\iota_1, \ldots, \iota_n\}$. We furthermore adopt the terminology of infochemicals (see Subsection 4.1.1), i. e. a pheromone is denoted as $\varphi \in \mathcal{P}$ with $\mathcal{P} \subset \mathcal{I}$, an allomone is denoted as $\alpha \in \mathcal{A}$ with $\mathcal{A} \subset \mathcal{I}$, a kairomone is denoted as $\kappa \in \mathcal{K}$ with $\mathcal{K} \subset \mathcal{I}$, and a synomone is denoted as $\varsigma \in \mathcal{S}$ with $\mathcal{S} \subset \mathcal{I}$. In contrast to the subcategories of allelochemicals, we do not explicitly model each subcategory of pheromones, i. e. (+,-), (-,+), (+,+) pheromones, but assume these subcategories to be included in $\mathcal{P}$. An agent, which is able to emit and perceive infochemicals, is defined as an infochemical-processing agent, extending the basic definition of an agent (see Definition 2.2).

Definition 4.2 (Infochemical-processing agent)
An infochemical-processing agent Ag^{inf} is defined as an extension of an agent Ag such that

$$Ag^{inf} = (Sit, Dat^{inf}, Act^{inf}, f_{Ag}^{inf}, \theta, emr)$$

where

- Sit is the set of situations Ag^{inf} can be in
- Dat^{inf} is the extension of Dat with infochemicals
- Act^{inf} is the extension of Act with the actions
  - *emit* that emits an infochemical to the current locations of Ag^{inf}
  - *perceive* that perceives all infochemicals stored at the current location of Ag^{inf}
- f_{Ag}^{inf} is the extension of f_{Ag} for processing infochemicals
- θ is the type of Ag^{inf}

- *emr* is the emission rate (W in the natural context), by which Ag^{inf} regularly emits infochemicals

The explicit extension of an agent Ag by a type allows for the precise coordination of homogeneous and heterogeneous types of agents within the same coordination media and mechanism. The types may be hierarchically composed to higher types, reflecting the taxonomic ranks in biology, as well as being linked to other types, reflecting interspecific relationships in biology.

4.2.2 Coordination Media

In contrast to biology, in this formal coordination model we assume space to be finite as well as discrete and time to be infinite as well as discrete ($t \in \mathbb{N}$). Instead of 'point in time', we will furthermore use the more specific term *iteration*, indicating a discrete succession of points of time or events that can be numbered starting from 0 respectively 1. Thus, an iteration represents a synchronized point in time. This does not limit the applicability of the coordination model, because a succession of iterations can easily be constructed from a conventional time measure, e. g. by transforming time information to milliseconds since the start of the system.

The coordination media (space) in infochemical-based coordination is represented by an infochemical environment, which enhances the execution infrastructure of the agents, providing them with an active environment where they may share information.

Definition 4.3 (Infochemical Environment)
An infochemical environment Env^{inf} is defined as

$$Env^{inf} = (L, C)$$

where

- $L = \{l_1, \ldots, l_n\}$ is a set of discrete locations
- $C \subseteq \{(l_i, l_j) | l_i, l_j \in L \ \wedge i \neq j\}$ is a set of directed connections

By this definition, Env^{inf} introduces a spatial structure to the MAS in which the agents may emit and perceive digital infochemicals at discrete locations. From the viewpoint of a location, a directed connection can be *inbound* or *outbound*.

Definition 4.4 (Inbound connection)
Let $l_i, l_j \in L$ and $(l_j, l_i) \in C$. Then (l_j, l_i) is called inbound connection of l_i.

Definition 4.5 (Outbound connection)
Let $l_i, l_j \in L$ and $(l_i, l_j) \in C$. Then (l_i, l_j) is called outbound connection of l_i.

Consequently, the set $\{(l_j, l_i)|(l_j, l_i)$ is inbound connection of $l_i\}$ is called *inbound connections* of l_i, whereas the set $\{(l_i, l_j)|(l_i, l_j)$ is outbound connection of $l_i\}$ is called *outbound connections* of l_i. Furthermore, $l_j \in L$ is called *inbound neighbor* of $l_i \in L$, if $\exists(l_j, l_i) \in C : ((l_j, l_i)$ is inbound connection of $l_i)$. $l_j \in L$ is called *outbound neighbor* of $l_i \in L$, if $\exists(l_i, l_j) \in C : ((l_i, l_j)$ is outbound connection of $l_i)$.

In order to store infochemicals at locations, while maintaining the information on the direction the infochemical was propagated from, an infochemical buffer is associated with each connection. An infochemical buffer is a data structure, able to store infochemicals of different types.

4.2.3 Coordination Laws

The action $emit(\iota) \in Act_{Ag}^{inf}$ performed by an agent Ag emits an infochemical ι at the current location of the agent into the infochemical environment Env^{inf}. The propagation of ι in Env^{inf}, in more detail between two locations in L, occurs based on a propagation function (see Definition 4.6).

Definition 4.6 (Propagation function)

$$prop : \mathcal{I}, L, L \rightarrow \mathcal{I}$$

The propagation function is only defined abstractly in the coordination model and has to be instantiated by a concrete coordination mechanism. The instantiation thereby has to consider ι's current concentration γ_ι, its threshold concentration γ^{thresh}, its diffusion coefficient δ_ι, as well as optionally an propagation factor pf and a propagation rate pr defined for the type of ι. Thereby, $pf \in [0; 1]$ specifies the fraction of ι that is propagated equally among all the neighbors of a location. pr specifies the rate (counted in iterations) by which ι is propagated. The factor by which the concentration changes due to a propagation between two locations depends on the used coordination mechanism. However, ι has only a chance of being stored at a neighboring location, if its concentration is still above its threshold concentration, i. e. $\gamma_\iota \geq \gamma_\iota^{thresh}$.

Assuming that the concentration of ι is still above its threshold concentration, an aggregation function (see Definition 4.7) defines the storing of ι at a location in L in consideration of already present infochemicals. Again, the instantiation of this abstract aggregation function depends on a concrete coordination mechanism as well.

Definition 4.7 (Aggregation function)

$$aggr : \mathcal{I}, \mathcal{I}, L \rightarrow \mathcal{I}$$

An evaporation function (see Definition 4.8) defines the evaporation of ι stored at an location in L. Its instantiation similarly depends on a concrete coordination mechanism and has to consider γ_ι, γ^{thresh}, δ_ι, as well as optionally an evaporation

factor ef and an evaporation rate er defined for the type of ι. $ef \in [0;1]$ specifies the fraction of ι that remains after an evaporation. er specifies the rate (counted in iterations) by which ι is evaporated.

Definition 4.8 (Evaporation function)

$$evap : \mathcal{I}, L \rightarrow \mathcal{I}$$

Any concrete instantiations of the above three functions will depend on the application domain at hand as well as the solution to realize. This allows for functions that may model the biological transmission of infochemicals (see Equation 4.1) as well as any other propagation, aggregation, and evaporation behavior. The action $perceive \in Act_{Ag}^{inf}$ performed by an agent Ag^{inf} provides the agent with a view of all infochemicals stored at the current location of Ag. However, the infochemicals remain stored at the location and are not influenced by this action. In particular, an agent Ag^{inf} is not able to remove any infochemicals from Env^{inf}.

4.3 Design Pattern

For a more systematical engineering of efficient self-organizing emergent MASs based on the principles of infochemical coordination (see Section 4.1) and the adopted coordination model (see Section 4.2) it is important to capture both in a way that is more intuitively to software engineers and easily transferable for autonomous solutions to their problems. In software engineering, this is mostly done by structured design patterns that can be instantiated and used by the engineers according to the specific needs of their problems. Patterns in general were first introduced by Alexander [Ale77] for architectural design in 1977. The concept of design patterns in computer science has become popular in 1994 with the object-oriented paradigm [GHJV94]. In its most general sense, a design pattern is a *"recurring solution to a standard problem"* [SJF96]. The use of design patterns offers several advantages such as reducing design-time by exploiting off-the-shelf solutions, promoting collaboration by providing a shared ontology, and lowering the number of errors in the development, to name a few.

Design patterns have also found their way into the engineering of MASs. Early contributions in this area were mostly concerned with the life-cycle of agents, providing solutions related to resource access, mobility, and basic social skills, see e. g. [AL98, KKPS98, DWK01], but primarily did not focus on the engineering of self-organizing emergent MASs. However, after defining a first taxonomy for various self-* properties in self-organizing emergent systems [DH07a], De Wolf and Holvoet [DH07b, DH06] started to describe design patterns also for decentralized coordination mechanisms such as market-based coordination (see Subsection 3.3.1), tag-based coordination (see Subsection 3.3.3), token-based coordination (see Subsection 3.3.4), pheromone-based coordination (see Subsection 3.3.6), or field-based coordination (see Subsection 3.3.7). On a higher level of granularity, Gardelli et al. [GVO07]

have described basic patterns common to various biological systems, such as *replication*, *collective sort*, *evaporation*, *aggregation*, and *diffusion*. On a similar level of abstraction, Babaoglu et al. [BCD+06] have described further patterns of biological coordination, including *plain diffusion*, *replication*, *stigmergy*, *chemotaxis*, and *reaction-diffusion*. A proper combination of some of these patterns may produce more complex patterns for self-organizing emergent systems, e. g. in the case of stigmergy the basic patterns are evaporation, aggregation, and diffusion (propagation). In [GNO+08] a so-called *organic design pattern*[3] is presented for self-* systems that consist of a set of independent agents interacting with each other and where reconfiguration/adaptation can be expressed as a reallocation of roles.

An important issue that has to be taken into consideration for the capturing of the principles of infochemical coordination in a design pattern is, however, that only a very few people are software engineers and biologists together, aware of the complex meanings of the different infochemical types that may be used for an effective and efficient coordination. Thus, to promote the usage of the design pattern, again meaningful abstractions have to be made that allow on the one hand of course biologically-inspired instantiations of this pattern, but on the other hand also instantiations apart from biological background knowledge using neutral terms.

Although there exist different formats and structures for describing design patterns (see e. g. [GHJV94, Lin03, CSC04]), it is generally agreed that the following sections are mandatory [MD97]: A *pattern name*, providing a clear, distinguishable identifier for the pattern. A *context* section, describing a situation when the pattern would apply. A *problem* section, giving a precise statement of the problem to be solved, in this case several problem characteristics. A *forces* section, describing items that influence the decision for the pattern, indicating trade-offs that might be made. A *solution* section, describing how the problem is being solved, balancing the forces. For a more detailed description, we additionally add three optional sections: A *rationale* section, explaining why the solution is appropriate for the problem along with its achieved (self-*) properties. An *examples/known uses* section, presenting a non-exhaustive list of examples/references that illustrate the application of the pattern. A *related patterns* section, mentioning other decentralized coordination mechanisms that may be also of interest for the solution of the problem. Because this format is well known in software engineering, e. g. [GHJV94] uses a similar format, the usage of the pattern is promoted.

4.3.1 Context

The problem to be solved demands an autonomous solution that requires the decentralized coordination of multiple homogeneous and/or heterogeneous, more or less autonomous elements in order to achieve a common and globally coherent goal. The elements are situated in a physical or logical environment, which can be extended with an appropriate infrastructure, whereas the environment structure may

[3] Although labeled as pattern, the organic design pattern is rather a conceptual model, as a structured description of the model is not available

represent a part of or even the entire problem to be solved. Some kind of spatial movement of the elements may be required or information about the spatial location of the elements has to be exchanged. The only possible way to coordinate are local estimates of global information. The desired solution has to be robust, flexible, and scalable in the face of frequent dynamic changes in the environment or the system.

4.3.2 Problem

The problem to be solved is characterized by the following items:

- **Spatial routing:** Autonomous elements have to move or route themselves adaptively and as optimal as possible through the environment or problem structure. Elements may have to be attracted to certain locations or in a certain direction and be deterred from certain locations or directions, respectively.
- **Spatial awareness:** Autonomous elements have to be provided with abstract, simple, yet effective contextual information, i. e. spatial information such as distance and/or direction to a location, facilitating the coordination process.
- **Homogeneity and heterogeneity:** Homogeneous and heterogeneous autonomous elements with different capabilities regarding their mobility, ability to communicate, or functionality have to be taken into account and are part of the problem or the solution.
- **Robustness and adaptiveness:** Autonomous elements have to move appropriately and achieve or maintain the globally coherent goal in face of dynamic changes in the environment, e. g. obstacles, failures, emerging/vanishing locations, emerging/vanishing pathways.
- **Openness and scalability:** Autonomous elements may leave or join the coordination process at any time and any location without affecting the overall performance negatively. In case of leaving, a graceful degradation is expected. In case of joining, a smooth and seamless integration is expected.
- **Various information sources:** Various sources can produce various types of information that have to be considered. The information has to be processed in a completely distributed and decentralized environment.

4.3.3 Forces

The decision for the pattern is influenced by the following items, which may indicate trade-offs that have to be made:

- **Centralization vs. decentralization:** In relation to a centralized approach, a decentralized approach usually causes a communication as well as coordination overhead, except the information to control the system is intrinsically

distributed or every element has almost global knowledge about the system state. However, the global state usually can not be obtained without any further assumptions or restrictions. In return, in very dynamic environments a decentralized approach has no bottleneck or single point of failure.

- **Optimality vs. robustness/flexibility:** In an adaptive approach without central means to optimize its efficiency an optimal solution to a problem can not be guaranteed. On the other hand, in face of frequent dynamic changes in the environment or in the system itself, a durable optimal solution does not exist at all. In these instances, a robust and flexible approach may be preferable to an optimal but inflexible approach.

- **Exploration vs. exploitation:** In contrast to only exploit already known information, new information has to be explored sufficiently in order to have an adaptive solution. This prevents the autonomous elements from being trapped in local optima and supports finding new pathways or sources. On the other hand, a too high level of exploration may result in inefficient solutions.

- **Responsibility of the environment vs. the elements:** Effective coordination often requires intensive information processing and communication, which can be accomplished by the elements themselves or the environment they are situated in. In the former case, the elements may explicitly reason about the information and control which and when information is distributed. However, this may require complex reasoning algorithms and communication capabilities and is not recommended in dynamic environments. If in contrast the environment itself represents the needed coordination information by transparently processing and distributing it toward the elements, the elements will be able to use that information as a kind of "red carpet", which, when followed, achieves the global goals and avoids complex processing within the elements. Such coordination can be more dynamic and adaptive.

- **Greediness vs. purposefulness:** In decentralized approaches, the need for adaptive and flexible coordination usually rules out globally informed and purposeful decisions by the autonomous elements. Thus, the elements act "greedily" and try to exploit any information immediately, instead of disregarding some information in order to receive a greater benefit later.

4.3.4 Solution

The description of the appropriate solution to the problem, balancing the aforementioned forces, is manifold. For a more detailed structure, this section is subdivided into a *conceptual description* of the solution, a *parameter tuning* subsection describing the essential parameters that can be tuned in this solution, and an *infrastructure* subsection that describes the functionality that is required from an infrastructure to realize the solution.

4.3.4.1 Conceptual Description

The solution is inspired by the principles behind infochemical-based coordination in nature (see Section 4.1). To make these principles usable for decentralized coordination in computer systems, they have to be meaningfully adopted into a computational model. Figure 4.2 illustrates the description of this model by an UML diagram. In this model, a living organism is seen as an autonomous `Agent`, situated in a spatial `Environment` consisting of multiple `Location`s the agent may be situated on. Connections between the locations define the possible ways an agent may choose from in order to move between the locations, whereupon the connections may be directed or undirected as well as of different length, depending on the physical conditions.

An agent belongs to at least one `Type`, which in turn may be hierarchically composed to higher types, reflecting the taxonomic ranks in biology, as well as being linked to other types, reflecting interspecific relationships in biology. This allows for homogeneous as well as heterogeneous agents situated in the same environment interacting with each other. An agent acting as `emitter` is able to emit digital `Infochemical`s, i. e. [+,-], [-,+], or [+,+] `Intra-type` infochemicals respectively `Inter-type` infochemicals, according to a specific `emission rate` into the environment, in order to communicate and coordinate with other agents indirectly. The abstraction of the biological terms pheromones and allelochemicals in the model by the terms intra-type and inter-type infochemicals allows on the one hand still the use of certain pheromone or allelochemical types in biologically-inspired instantiations of this pattern, but on the other hand also instantiations apart from biolocigal background knowledge using neutral terms. A digital infochemical basically contains four attributes:

- `Individual information`, which reflects the biological role of an infochemical as a dynamic information carrier. The content of this attribute may vary between different applications, whereas at least the type of the emitting agent is included in this information.
- Its `current concentration`, which reflects the dynamically changing concentration of diffusing biological infochemicals.
- A `threshold concentration`, which reflects the behavioral threshold concentration of living organisms regarding specific infochemicals.
- Its `diffusion coefficient`, which reflects its biological pendant and thus allows for a very fine-tuned propagation radius and evaporation time specific to each infochemical.

An agent emits an infochemical to the environment by handing it over to the location it is currently situated on. All locations in the environment are in charge of providing a stigmergic functionality to the agents. So every location is able

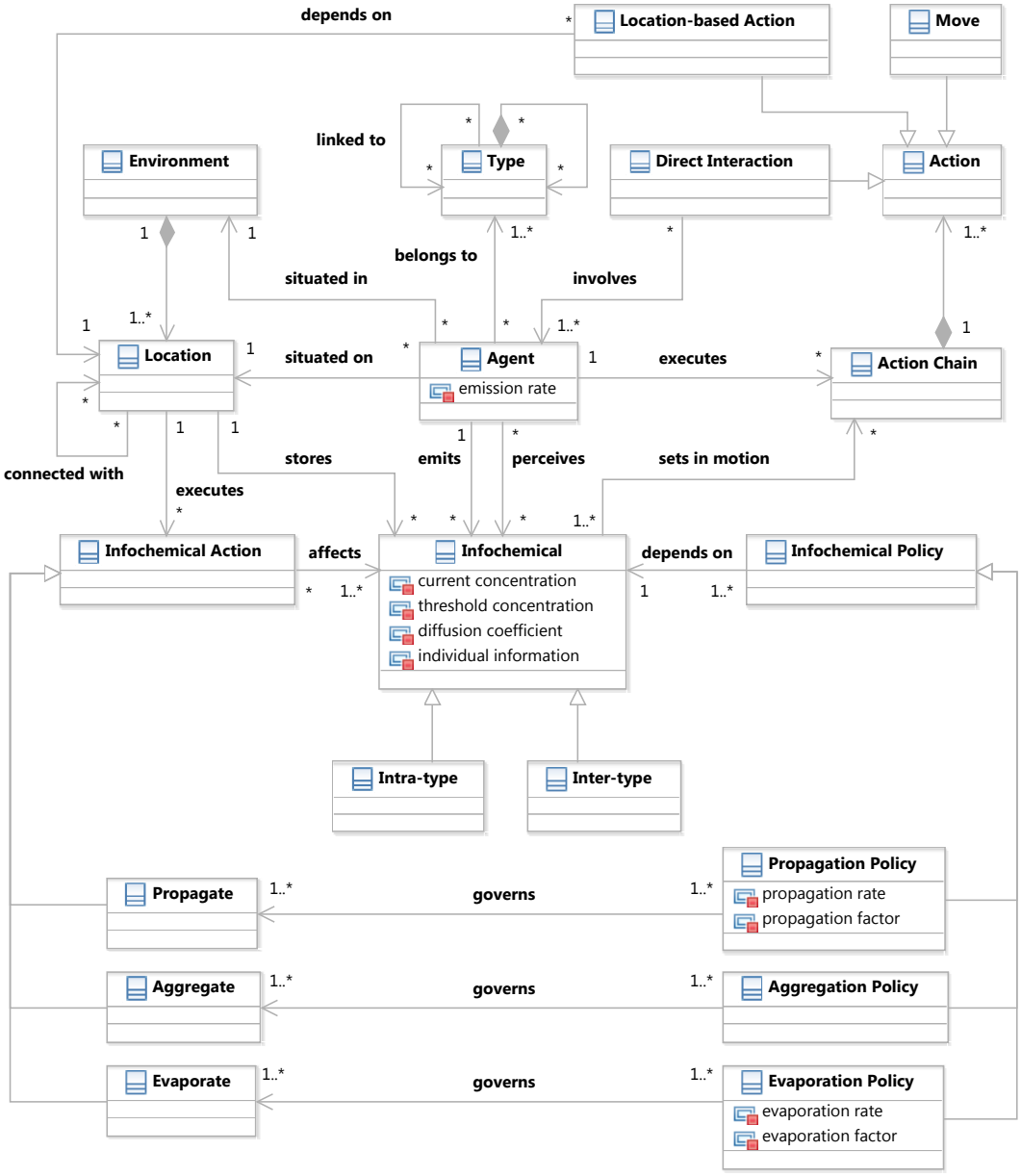


Figure 4.2: Conceptual model of DIC

to execute three different `Infochemical Action`s, each governed by a respective `Infochemical Policy`:

1. A location may `propagate` an infochemical to its neighboring locations according to an infochemical-specific `Propagation Policy`. The amount that is propagated is governed by a `propagation factor` and the infochemical-specific diffusion coeffcient, both affecting the decrease of the infochemical concentration. The rate is governed by a `propagation rate`. Propagation as such supports information diffusion and spreading.

2. A location may `aggregate` different infochemicals according to an infochemical-specific `Aggregation Policy`, such that separate infochemicals are perceived as one. Aggregation in general is a mechanism of reinforcement and supports information fusion.

3. A location may `evaporate` infochemicals according to an infochemical-specific `Evaporation Policy`. An individual `evaporation rate` governs the speed of evaporation, whereas the infochemical's diffusion coefficient as well as its `evaporation factor` govern the amount that is evaporated. Evaporation in general serves to forget old information that is not refreshed or reinforced by new infochemicals, which supports truth maintenance of information in the environment.

Due to these infochemical actions and policies an infochemical emitted by an agent diffuses across the neighboring locations in the environment. Every location affected by this diffusion stores a certain quantity of the infochemical, as long as the infochemical has not to be removed. The diffusion will produce a kind of infochemical field around the emitting agent. The location with the highest infochemical concentration of a field is the one the emitting agent is currently situated on.

An agent acting as `perceiver` is able to perceive infochemicals contained at its current location, possibly emitted by itself, by agents of its own type, or by agents of another type, in case that the relevant types are linked together. The perception of an infochemical by an agent may set in motion an `Action Chain` executed by the agent, reflecting the individual function of a given infochemical on a living organism. An action chain consists of at least one `Action`, which can be of the following types:

- `Move`: The agent moves from its current location to a neighboring location, depending on the perceived infochemical field. The movement can be in the direction of the perceived infochemical, in the opposite direction, or equal to the concentration of the infochemical field, depending on the desired behavior of the agent and the coordination to achieve.

- `Location-specific Action`: This action can have different shapes. On the one hand, triggered by the perception of e. g. an alarm pheromone, an agent may response by emitting alarm pheromones in turn. On the other hand, if

its current location has a special meaning to the agent, the agent may also execute a well-defined action at this location. For example, an ant picks up food at its destination location and drops food at its source location, while it emits pheromones at every location in between.

- `Direct Interaction`: The agent directly interacts with one or more other agents. Direct interaction is only possible, if the interacting agents are situated on the same location. The reasons for direct interactions can e. g. be information exchange, reflecting the direct exchange of pheromones between ants or between bees in the case of surface pheromones, or resource exchange, reflecting the exchange of resources, e. g. pollen grains between flowers and bees for instance, but also any other act of communication or negotiation.

Independent of a perceived infochemical triggering a reactive action, an agent may also execute certain action chains proactively, e. g. if no infochemicals can be perceived on the current location or infochemicals are to be emitted due to other reasons.

4.3.4.2 Parameter Tuning

The parameters that may be tuned in IBC for an efficient coordination are the following:

- **Emission concentration:** The initial concentration of an infochemical when it is created by an agent and emitted into the environment. Even though more than one type of infochemical is used for the coordination, the emission concentration value applies to all types.
- **Threshold concentration:** The minimal concentration of an infochemical. If the concentration of an infochemical falls below this value due to its propagation and evaporation, the infochemical will be removed from the environment. If there are different types of infochemicals used for the coordination, every infochemical type may have its own threshold concentration. A higher threshold concentration value narrows the diffusion area of an infochemical, a lower threshold concentration widens the area.
- **Emission rate:** The rate by which an infochemical is emitted by an agent. If there are different types of agents participating in the coordination process, every agent type may have its own emission rate. A high emission rate should be used, if information changes frequently or other agents have to be noticed of information changes, a lower emission rate should be used, if information is rather static.
- **Diffusion coefficient:** If there are different types of infochemicals used for the coordination, every infochemical type may have its own diffusion coefficient, allowing for fine-tuned information diffusion areas. While some information

may be required to be spread over a greater distance, other is not. Thus, an unnecessary communication overhead can be reduced. Note, agents could also be allowed to change the diffusion coefficient of their emitted infochemicals dynamically, in order to adapt to possible changed situations.

- **Propagation factor/rate** and **evaporation factor/rate:** Together with the diffusion coefficients, these parameters control the information spreading and truth maintenance individually for each participating type of infochemical. The settings may depend on the application-specific propagation/evaporation functions, which may be linear, degressive, distance-dependent, etc. In general, if evaporation proceeds very fast, information will be forgotten more rapidly. If evaporation is too slow, too many agents will be attracted into the wrong direction.

A proper tuning of these parameters has significant impact on the efficiency of the system. However, it is recommended to keep the amount of variable parameters as small as is necessary.

4.3.4.3 Infrastructure

The application of this solution requires a kind of *infochemical infrastructure* to be provided by the locations composing the environment. Every location therefore has to provide a certain functionality:

- It has to accept and store infochemicals emitted by an agent situated on it.
- It has to propagate, aggregate, and evaporate infochemicals according to the respective infochemical policies.
- It has to provide access to locally stored infochemicals for an agent situated on it.

The realization of this infrastructure usually depends on the application domain. In case of a MAS running on a single machine, for example, the implementation will be simply a piece of software. In case of a distributed manufacturing system, for example, a kind of distributed middleware will be required.

4.3.5 Rationale

The appropriateness of the solution along with its achieved (self-*) properties is explained by the following items:

- **Routing:** In general, following the increasing concentration of an infochemical field is the shortest path to the emitter. Attracting agents to specific locations and to move in a specific direction according to an infochemical's concentration is supported as well as repelling an agent from a location or direction. Obstacles are bypassed adaptively.

- **Feedback:** Feedback is given by the fact that infochemicals can change when changes occur in the environment, when the agent that emits the infochemical decides to move, or is required to change the individual information, which is included in the infochemical. Other agents can then take the perceived change into account and react on it by, for example, emitting corresponding infochemicals on their own or changing the individual information of their own infochemicals according to the observed information (positive feedback). As outdated information is not refreshed anymore and gradually evaporates, negative feedback occurs. As such feedback cycles are established enabling self-organization.

- **Environment topology:** The structure of the environment reflects a part or even the entire problem to be solved. The distribution of the infochemicals along with their concentration guides the agents to the current solution of that problem.

- **Decentralized control:** Local decisions are made without requiring centralized reasoning or control. This way a global self-organized motion pattern emerges due to the related effects of agents emitting infochemicals and moving according to other observed infochemicals. The goals that are accomplished are not due to single agents, but due to the system as a whole without any central controller.

- **Information diversity:** Various types of information from various sources are supported. Coordination can be based on multiple types of infochemicals, even on multiple types of the same infochemical type, e. g. different types of pheromones.

- **Dynamic situations:** The environment in general is able to incorporate dynamic changes immediately, enabling the agents to react in a *flexible* way. New information is quickly integrated, while outdated information is quickly forgotten. Concepts such as exploration, information refreshment, and evaporation result in an *adaptive* coordination process. Agents thus can join and leave the system without significant disturbances to the global goal. This *openness* of the coordination process makes the mechanism extremely *robust*. Due to the intrinsically decentralization, the entire mechanism is *scalable* in problem size.

- **Information spreading/distribution:** Infochemicals are dynamic information carriers holding spatial information (direction to or distance to emitter) as well as individual information. The environment represents the distribution mechanism for this information and participates actively in the system's dynamics.

- **Processing complexity:** The agents are responsible for which information is emitted where and when into the environment. The environment is then responsible for storing, propagating, and evaporating this information. As such,

the environment makes sure that not too much computational and communication burden is imposed on the agents themselves by automatically providing a dynamically adapting and propagating coordination structure that is immediately usable by agents. The context is represented expressively as infochemical fields, i. e. a kind of "red carpet", which represents how to achieve a coordination task by simply following the field. The coordination is achieved with very little effort and without complex reasoning by the agents. The latter indicates that the problem solving power resides in the local interactions instead of inside the agents' reasoning.

- **Self-* properties:** According to the characteristics of self-* properties in decentralized AC systems [DH07a], the solution usually achieves smoothly evolving, ongoing, macroscopic, and adaptation-related self-* properties with possible functionalities in resource allocation, group formation, spatial shaping, or load balancing, to name a few. The inherent adaptiveness, flexibility, and robustness of the coordination process yield to some extend self-configuring and self-healing properties.

4.3.6 Examples/known uses

Examples, known uses, and references of the DIC pattern can be found in different problem domains. A non-exhaustive list illustrating the application of the pattern includes the following:

- Multiple types of pheromones with varying propagation rates and thresholds have been used in [BP00] to coordinate agents of two species on a hexagonal grid in a military scenario. Evaluations have verified the performance improvements that were achieved due to the different pheromone configurations. Similarly, in [PL04] multiple types of pheromones have been used for self-optimizing trail behaviors by agents in domains which have obstacles, dynamically changing target locations, and multiple waypoints.

- In [KBD08] a self-organizing emergent system was developed for the solution of Pickup and Delivery Problems (PDPs) in manufacturing systems. It was demonstrated that a combination of both intraspecific and interspecific interactions by different types of chemicals in the same system yields more powerful and efficient solutions. Experiments executed on the same system in [KDB09] show that infochemical-specific diffusion coefficients along with individual propagation and evaporation policies result in a significant message reduction with a simultaneous performance increase.

- Based on the DIC pattern, in [DDKB10] a self-organizing emergent system was developed for the real-time control of water distribution networks. Tanks and pumps of water distribution networks were equipped with infochemical-processing agents that coordinate by means of two to three different infochem-

icals. The decision making in the agents based on these infochemicals was able to outperform human decision making in certain situations.

- The idea of dropping information on specific locations that are picked up by other agents can even be found in search problems. In [YK96] a state-space search problem is solved concurrently by multiple cooperative agents, whereby the agents exchange information they found during their search, which can then be used by other agents arriving at these *logical* locations.

4.3.7 Related patterns

IBC is naturally related to other mechanisms in charge of coordinating multiple autonomous agents in a self-organizing manner, in particular field-based coordination (see Subsection 3.3.7) and of course pheromone-based coordination (see Subsection 3.3.6). Field-based coordination is similarly to IBC an instantiation of classical gradient field-based coordination, but inspired from magnetic fields. However, there the gradient parts do not evaporate over time but have to be removed explicitly by the environment. The strength of the gradient parts usually increases with increasing distance to the gradient initiator, which sometimes leads to the problem of local minima when gradient fields are combined for the coordination of agents. Pheromone-based coordination can be readily considered as a specialization of IBC, as it supports only the coordination of homogeneous autonomous agents by means of digital pheromones. Also, there all pheromones are propagated and evaporated equally without the possibility to differentiate between various types, which limits its general applicability.

4.4 Design Guidelines

Once an engineer has determined that the IBC design pattern presented in the last section is applicable to his particular situation as well as to its desired self-* properties, the next step is to find respectively design an appropriate decentralized coordination mechanism that instantiates the pattern respectively the coordination model. If such a mechanism has not be designed for the problem in hand yet, the engineer consequently has to design a new one. However, without any further support this process may be complex, time-consuming, and costly again, because the appropriate utilization and combination of different types of infochemicals along with their inter-related effects on the local (microscopic) as well as on the global (macroscopic) level is hard to understand due to the micro-macro gap.

Therefore, as a further artifact, in this section we present design guidelines that support engineers in specifying new coordination mechanisms based on IBC. More precisely, the guidelines provide an indication which types of infochemicals to use for the agents' interactions on the local level in order to achieve certain required effects on the global level. Fur this purpose, we again take advantage of biology: Any local interactions between living organisms or species in an ecosystem establish

a global relationship between the interacting organisms. These relationships can be considered as macroscopic effects emerging from the microscopic behavior, i. e. the emitting of infochemical types, of the organisms.

As mentioned in Section 4.1, one distinguishes two different types of interactions between living organisms: intraspecific interactions between organisms of the same species and interspecific interactions between organisms of different species. Both types result in different relationships between the interacting organisms and have either beneficial, neutral, or harmful effects on the organisms or species involved. The level of benefit or harm however is continuous and not discrete [Lid79]. *Intraspecific* relationships can be classified along the degree of sociality exposed, ranging from nonsocial to social:

- *Nonsocial relationships* emerge from the tendency of organisms to avoid the association with others and to disregard the welfare of the own species. The reason may be a negative or positive egoistic behavior of an organism. *Negative egoistic behavior* refers to forms of selfish interactions, if the outcome is harmful to the general welfare of the own species or detrimental to other organisms. *Positive egoistic behavior* instead refers to forms of selfish interactions, if the outcome is self-interested but not harmful to the general welfare of the own species or detrimental to other organisms.

- *Social relationships* emerge from the tendency of organisms to associate with others and to form social groups. The reason may be a reciprocal altruistic, altruistic, or selfless behavior of an organism. *Reciprocal altruistic behavior* refers to the conversion of egoistic interactions into altruistic interactions. It describes social or cooperative forms of interactions, in which one organism provides a benefit to another organism of the same species without expecting any payment or compensation immediately ("tit for tat"). For example, vampire bats feed regurgitated blood to those who have not collected much blood themselves, having in mind that they themselves may someday benefit from this same donation. *Altruistic behavior* refers to social forms of interactions that increase the fitness of another organism of the same species without any immediate or later increase of the own fitness. For example, dolphins support sick or injured animals, swimming under them for hours at a time and pushing them to the surface so they can breathe. *Selfless behavior* refers to social forms of interactions that increase the fitness of another organism while decreasing the own fitness. The organism therefore follows a course of action that has a high risk or certainty of suffering or death, which could otherwise be avoided. For example, vervet monkeys give alarm calls to warn fellow monkeys of the presence of predators, even though in doing so they attract attention to themselves, increasing their personal chance of being attacked.

Having these relationships in mind, the guidelines provided in Table 4.2 can be read in two ways: If an engineer knows about the macroscopic intraspecific relationships he wants to achieve, he may look up the corresponding pheromone types

that have to be used for the interactions between homogeneous agents (agents of the same type) on the local level therefore. If an engineer by contrast knows about the microscopic effects he wants to achieve between homogeneous agents, he may look up the corresponding pheromone types as well. The effects on the interacting agents are denoted by '+' (positive case), '0' (neutral case), or '−' (negative case). The meaning of these effects depends on the addressed problem domain, e. g. for solutions that require a spacial movement or routing of agents a '+' may indicate an attraction toward the direction of the perceived pheromone, whereas a '−' may indicate a repellent effect into the opposite direction, provided that the microscopic behavior of the agents reflects the corresponding biological behavior of the organisms.

Macroscopic intraspecific relationship	Effect on source of stimulus	Effect on sink of stimulus	Corresponding type of pheromone
Nonsocial			
- Negative egoistic	+	-	Territorial pheromone
- Positive egoistic	+	0	Home range pheromone
Social			
- Reciprocal altruistic	+	+	Dispersal, trail, sex, surface pheromone
- Altruistic	0	+	Aggregation pheromone
- Selfless	-	+	Alarm pheromone

Table 4.2: Macroscopic intraspecific relationships with corresponding microscopic effects and pheromone types

Interspecific relationships that emerge between organisms or populations of different species range from antagonistic to mutualistic:

- *Antagonistic relationships* emerge from the interactions between (organisms of) different species, in which one species benefits at the expense of another species. There are two types of antagonism: predation and parasitism. *Predation* describes a relationship where a predator organism feeds on another living organism. *Parasitism* describes a relationship, in which one species, the parasite, benefits from the interaction with the other species, the host, which is harmed.

- *Amensal relationships* emerge from the interactions between (organisms of) different species, in which one species impedes or restricts the success of the other without being affected positively or negatively by the other. Amensalism sometimes is further divided into *competition* and *antibiosis*. For example some higher plants, e. g. the already mentioned black walnut, secrete substances, e. g. juglone, that inhibit the growth of – or kill outright – nearby competing plants.

- *Commensal relationships* emerge from the interactions between (organisms of) different species, in which one species benefits and the other species is neither benefited nor harmed. For example, whereas a small crab enters the shell of a oyster as a larva and receives shelter while it grows, once fully grown it is unable to exit through the narrow opening of the two valves and remains within the shell, snatching particles of food from the oyster but not harming its unwitting benefactor.
- *Mutualistic relationships* emerge from cooperative interactions between (organisms of) different species that normally benefit both species. They can be thought of as a form of biological barter in which species trade resources, such as carbohydrates or inorganic compounds, or services, such as protection from predators. For example, plants are hosts for insects that visit and pollinate them or eat their fruit.

Analogously to Table 4.2, the guidelines provided in Table 4.3 can be read in the same two ways, however for the interactions between heterogeneous agents (agents of different types).

Macroscopic interspecific relationship	Effect on source of stimulus	Effect on sink of stimulus	Corresponding type of allelochemical
Antagonistic			
- Predatory	+	-	Foraging, enemy-avoidance allomone
- Parasitic	-	+	Foraging, enemy-avoidance kairomone
Amensal			
- Competitive/antibiotic	0	-	Allomone (in general)
Commensal			
- Inwards directed	+	0	Sexual allomone
- Outwards directed	0	+	Sexual, aggregation kairomone
Mutualistic			
- Resource-resource coupled	+	+	Synomone
- Service-resource coupled	+	+	Synomone
- Service-service coupled	+	+	Synomone

Table 4.3: Macroscopic interspecific relationships with corresponding microscopic effects and allelochemical types

There also exist relationships that apply to both types of interactions, in which the fitness of one organism or species has absolutely no effect on that of the other organism or species (*neutral relationships*) or the fitness of both is decreased (termed

synnecrosis). However, true neutralism is very unlikely and impossible to prove, whereas synnecrosis is a rare and necessarily short-lived relationship as evolution selects against it.

4.5 Exemplary Instantiation

As mentioned in Subsection 4.3.7, pheromone-based coordination (PBC) can be considered as a specialization respectively instantiation of the DIC model, as it supports the coordination of homogeneous agents by means of one single type of digital infochemical, namely pheromones. In order to show that the specification of the DIC model generalizes the specification of PBC, as e. g. used in [Brü00, DS04, HVKB04, SH05a, VHG+07], in this section we exemplary instantiate the DIC model for this specific decentralized coordination mechanism in its original formulation. We will use the same specification structure later for the instantiation of the DIC model for PIC (see Subsection 7.2.2, later).

4.5.1 Biological Inspiration: Ant Foraging

Although many species in nature coordinate their local activities indirectly by means of pheromones (e. g. ants, bees, termites, etc.), PBC was inspired by the stigmergic communication of ants in the first instance. To be more specific, because ants utilize different shapes of marker-based and sematectonic stigmergy (see Subsection 2.3.2.1), e. g. foraging, division of labor, brood sorting, and cooperative transport, PBC is inspired by the local behavior of ants during foraging.

Foraging in social ant colonies, such as the *Lasius niger* or the Argentine ant *Iridomyrmex humilis*, is based on pheromones rather than on visual cues [GADP89, HW90]. Thereby, the local behavior of a single ant is very simple. At its current location, the ant observes the local concentration and the local gradient of the concentration of pheromones. Due to the releaser effect of pheromones, in this case trail pheromones (see Subsection 4.1.2), the ant changes its behavior, i. e. it tends to walk, probabilistically, in the direction with the highest concentration of pheromones. When the ant finds a food source, it deposits a certain amount of pheromones on the ground itself, while it heads back to the nest. Thereby, multiple deposits of a pheromone at the same location aggregate in strength. Other ants that perceive this pheromone trail then similarly tend to walk, probabilistically, in the direction with the highest concentration of pheromones. If they still find food, they will similarly head back to the nest, while depositing own pheromones, which again reinforce the trail. The more food is found, the more ants will follow the trail, the stronger the concentration of the trail.

The trail thus emerges as the shortest path between the nest and the food source solely from the actions and interactions of the individual ants. Even if obstacles such as rocks or pieces of wood change the environmental conditions, the ants will be able to find their way and to maintain the pheromone trail (see [DAGP90]).

Consequently, once the food source is exhausted, the trail will not be reinforced but will disappear due to the evaporation of the pheromones.

4.5.2 Pheromone-based Coordination

To specify PBC, in Subsection 4.5.2.1 we first describe the conceptual model of PBC as an instance of the IBC approach. Subsection 4.5.2.2 subsequently specifies the objective coordination in PBC, i. e. the coordination between the agents, whereas Subsection 4.5.2.3 specifies the subjective coordination in PBC, i. e. the coordination within an agent.

4.5.2.1 Conceptual Model

Based on the aforementioned inspiration, Figure 4.3 depicts a conceptual model of the PBC mechanism as an instantiation of the DIC model (see Figure 4.2). An `Agent` in PBC, which represents an ant in nature, instantiates an *agent* of the DIC model and is situated on a `Location` part of an `Environment` the agent is situated in. An agent can emit `Trail Pheromone`s and perceive pheromones emitted by other agents or itself. The emission of a pheromone is always a `Location-based Action`. The pheromones are affected by `Infochemical Action`s executed by a location, in more detail the locations propagate, aggregate, and evaporate the pheromones. These actions are governed by respective `Infochemical Policy`s. Due to the pheromones stored by the locations, pheromone trails may emerge, which an agent may follow from its `Source` to a `Destination` and vice versa, where it can in turn perform a `Destination Action` respectively a `Source Action`.

4.5.2.2 Objective Coordination

In indirect coordination mechanisms such as PBC, the objective coordination, i. e. the coordination between the agents, is specified by defining the behavior of the coordination environment in response to a message, in this case the behavior of the infochemical environment in response to a pheromone. Therefore, we first specify a pheromone in PBC more formally. The behavior of the infochemical environment is then defined by specifying the behavior of a location part of the environment.

Trail Pheromone

A trail pheromone φ emitted by an agent Ag is specified as

$$\varphi = (\gamma, \gamma^{thresh}) \tag{4.2}$$

where

- $\gamma \in \Re$ is the concentration of φ
- $\gamma^{thresh} \in \Re$ is the threshold concentration of φ

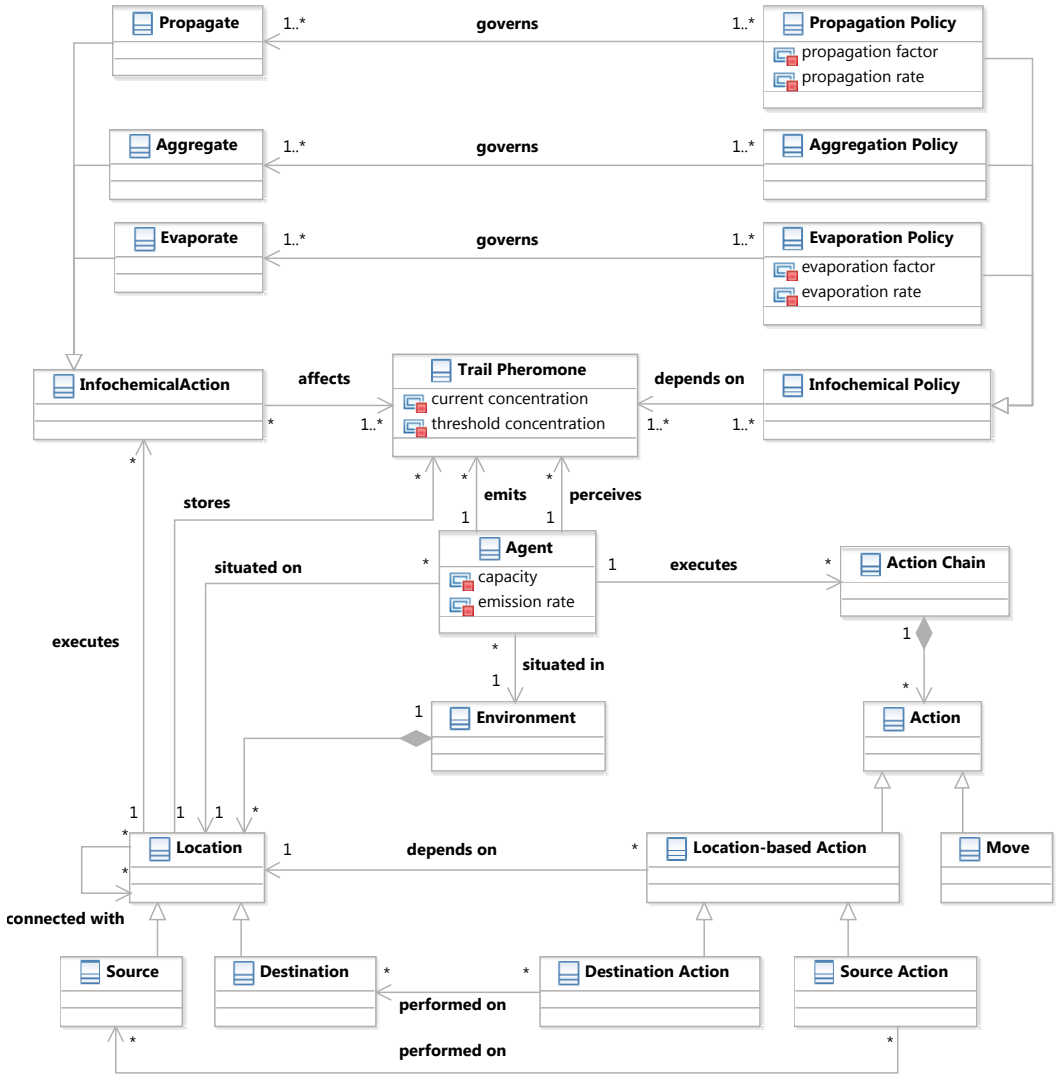


Figure 4.3: Conceptual model of PBC

In PBC, trail pheromones are usually not required to comprise any individual information. This is because PBC only utilizes quantitative stigermgy, so that only the amount (concentration) of pheromones at a certain location plays a role. By contrast, PIC (see later Subsection 7.2.2) utilizes quantitative and qualitative stigmergy for coordination, where individual information included in the infochemicals is essential.

Behavior of a Location

Every location part of the infochemical environment is able to execute three different actions affecting an infochemical. In more detail, a location can propagate, aggregate, and evaporate an infochemical, depending on specific infochemical policies. To specify these policies, we will instantiate the propagation, aggregation, and evaporation function defined by DIC.

Propagation Function Equation 4.3 instantiates the propagation function defined in Definition 4.6 for the PBC mechanism. While a pheromone φ is propagated from a location l_1 to a location l_2, its concentration is decreased depending on the distance $d_{l_1 l_2}$ between the two locations (measured in e. g. in meters, hops, etc.), the propagation factor pf, and the propagation rate pr of φ. If the concentration of φ at the location l_2 would be below the threshold concentration $\gamma_{\varphi}^{thresh}$ defined for pheromones, φ will not be stored at l_2.

$$prop : \varphi_{l_1} \mapsto \begin{cases} \varphi_{l_2} & \text{if } \gamma_{\varphi_{l_2}} \geq \gamma_{\varphi}^{thresh} \text{ where } \gamma_{\varphi_{l_2}} = \gamma_{\varphi_{l_1}} - d_{l_1 l_2} \cdot pf_{\varphi} \cdot pr_{\varphi} \\ \emptyset & \text{if } \gamma_{\varphi_{l_2}} < \gamma_{\varphi}^{thresh} \text{ where } \gamma_{\varphi_{l_2}} = \gamma_{\varphi_{l_1}} - d_{l_1 l_2} \cdot pf_{\varphi} \cdot pr_{\varphi} \end{cases} \tag{4.3}$$

Aggregation Function Equation 4.4 instantiates the aggregation function defined in Definition 4.7. In PBC, two pheromones φ_1 and φ_2 present at location l are aggregated to a single pheromone φ, whose concentration is the sum of the concentrations of φ_1 and φ_2.

$$aggr : \varphi_{1,l}, \varphi_{2,l} \mapsto \varphi_l \text{ with } \gamma_{\varphi_l} = \gamma_{\varphi_{1,l}} + \gamma_{\varphi_{2,l}} \tag{4.4}$$

Evaporation Function Equation 4.5 instantiates the evaporation function defined in Definition 4.8. In PBC, a pheromone φ stored at location l decreases its concentration γ_{φ} between iteration t and $t+1$ depending on the evaporation factor ef and the evaporation rate er defined for a pheromone. If the concentration of φ at iteration $t+1$ would be below the threshold concentration $\gamma_{\varphi}^{thresh}$ defined for pheromones, φ will be removed from l.

$$evap : \varphi_{t,l} \mapsto \begin{cases} \varphi_{t+1,l} & \text{if } \gamma_{\varphi_{t+1,l}} \geq \gamma_{\varphi}^{thresh} \text{ where } \gamma_{\varphi_{t+1,l}} = \gamma_{\varphi,l} \cdot ef_{\varphi} \cdot er_{\varphi} \\ \emptyset & \text{if } \gamma_{\varphi_{t+1,l}} < \gamma_{\varphi}^{thresh} \text{ where } \gamma_{\varphi_{t+1,l}} = \gamma_{\varphi,l} \cdot ef_{\varphi} \cdot er_{\varphi} \end{cases} \tag{4.5}$$

4.5.2.3 Subjective Coordination

The subjective coordination, i. e. the coordination within an agent, is specified by defining the simple local behavior of an agent Ag part of the PBC mechanism (see Algorithm 4.1). If Ag is currently not situated on a location but is moving between two locations on a connection, it will proceed moving to the next location (see lines 1–2). If Ag is situated on a location, which is its current destination, e. g. the food source, and it has not performed its destination action so far, e. g. pickup food, it will perform its destination action (see lines 3–4). If the location Ag is situated on is a location, which is its source, e. g. the nest it originated from, and Ag is returning from its destination as well as has not performed its source action so far, e. g. drop food, it will perform its source action (see lines 5–6). Otherwise, Ag perceives from every outbound connection the current pheromone strength (see lines 8–10) and moves based on a probabilistic or stochastic decision making in the direction of the pheromone with the strongest concentration (see line 12). If no pheromone is perceived, Ag chooses a random outbound connection to move on (see line 14). If Ag is returning from its destination, it will emit fixed amounts of pheromones at a constant emission rate (see lines 15–16).

Algorithm 4.1 Local behavior of an agent in PBC

```
    if l = ∅ then
2:      do proceed moving
    else if l = destination ∧ ¬ destination action performed then
4:      do destination action
    else if l = source ∧ returning ∧ ¬ source action performed then
6:      do source action
    else
8:      for all c_out ∈ l do
            do perceive φ
10:         P_perceived = P_perceived ∪ φ
        if P_perceived ≠ ∅ then
12:         do move on c_out of φ ∈ P_perceived with γ_φ = max
        else
14:         do move on random c_out
        if returning then
16:         do emit φ
```

4.6 Related Work

Because the DIC model explicitly allows for the specification of different functions, dynamics, and semantics of digital infochemicals as well as the usage of different types of agents all within one single decentralized coordination mechanism, it provides an enhanced expressiveness and efficiency in particular compared to the PBC mechanism as described in the last section. The experiments described in Section 8.3 will underpin this statement by resilient results.

Furthermore, the general biological principles of infochemical-based coordination in nature adapted by the DIC model provide an increased functionality that can be realized by one single coordination approach (see Table 4.4). For instance, by the PIC mechanism specified in Subsection 7.2.2 below, we will demonstrate an instantiation of the DIC model that uses *indirect interaction* to fulfill *resource allocation* as well as a *spatial distribution* of agents within one single coordination mechanism. Furthermore, due to the inclusion of individual information into digital infochemicals and their propagation via the environment, *information dissemination* is provided by the DIC model as well. The construction of *spatial shapes* is exemplary realized by the PBC mechanism, considered as an instantiation of the DIC model.

	Market-based	Gossip-based	Tag-based	Token-based	Immunity-based	Pheromone-based	Field-based	Infochemical-based
Resource allocation	+			+				+
Group formation			+					+
Organizations				+				
Self-protection			+	+	+			+
Information dissemination		+					+	+
Spatial shapes		+				+	+	+
Spatial distribution	+						+	+
Indirect interaction	(+)					+	+	+
Source of inspiration	(+)					+		+

Table 4.4: Comparison of decentralized coordination approaches

The DIC model in principle can be used for *group formation* as well, if the agents are allowed to change the type of other agents (similar to tag-based coordination). Also, similar as in nature, the DIC model provides a form of *self-protection*, as long as the agents only react on the perception of infochemicals emitted by agent types that are linked to the own agent type. However, these two criteria have not been experimentally tested, yet.

Due to the plethora of examples of IBC in biology, the DIC model inherently provides a nearly inexhaustible *source of inspiration* for the design of new decentralized coordination mechanism. Apart from PIC, an example is given by a DIC-based decentralized coordination mechanism, called *indirect defense coordination*, used for the realization of a self-organizing emergent MAS for the real-time control of water distribution systems [Döt10]. This mechanism is inspired by an infochemical-based strategy certain plants apply in biology in order to deter herbivores (see [DvPdB03]).

4.7 Conclusion

In this chapter, we have presented several design artifacts that can be used for the systematic engineering of effective and in particular more efficient self-organizing emergent MAS solutions. The artifacts simplify the design, save development time, and thus reduce development costs as well as TCO (see *Objective 1*).

In more detail, we have investigated the general principles behind infochemical-based coordination in biology and adopted them in the DIC model. On the one side, the DIC model thus allows for the efficient coordination of homogeneous and heterogeneous agent types by different types of infochemicals within one single, versatile, and coherent model (see *Challenge 1*). On the other side, the DIC model thus furthermore allows for the identification and specification of a vast amount of new and efficient, biologically-inspired coordination mechanisms (see e. g. later Subsection 7.2.2), which are based and guided by the expressiveness of the DIC model (see *Challenge 2*). In addition to the DIC model, we have provided a corresponding design pattern as well as design guidelines that help to identify and adapt decentralized coordination mechanisms (see *Challenge 3*).

Even though the design pattern and the design guidelines can not repudiate their biological origin, they do not force an engineer to be a biological expert and to understand the complicated relationships in nature. Once an engineer in charge of designing a self-organizing emergent MAS solution has identified an appropriate set of agents, which can even be done based on conventional agent-oriented software engineering (AOSE) techniques (see Section 3.4), the engineer first looks at the *context*, *problem*, and *solution* sections of the design pattern description. Once he has determined that the pattern is of interest, he looks at the *forces* and *rationale* sections for guidance on determining whether the pattern is applicable to his particular situation and to his desired self-* properties. The next step is to find respectively design an appropriate decentralized coordination mechanism that instantiates the design pattern respectively the DIC model. If such a mechanism has not been identified respectively designed for the problem in hand yet, the engineer consequently has to design a new one. Therefore, the design guidelines provide an indication, which types of infochemicals to use for the agents' interactions on the local level in order to achieve certain required effects on the global level. When using an iterative engineering process, as e. g. done in [KBD08], the design guidelines moreover help to adapt and improve the coordination mechanism in each iteration, as they support the engineer in adding or changing infochemicals or even the agents' reactions to certain infochemicals.

The experimental results described in Section 8.3 will demonstrate the enhanced efficiency that can be achieved when designing self-organizing emergent MASs by the artifacts described in this chapter. However, one has to be aware of the fact that when using decentralized coordination approaches for a solution to a dynamic optimization problem, no approach in general will be able to guarantee the optimality of the solution nor a required degree of efficiency during operation on its own. In other words, the agents in a self-organizing emergent MAS will always tend to make

'suboptimal' *local* decisions during operation that may result in *global* inefficiencies. In the next part of this thesis, we will describe and analyze reasons for this drawback in more detail and present an appropriate approach to overcome such inefficiencies during operation.

Part III

Operation Phase

Chapter 5

Operating Self-Organizing Emergent Systems

While in the previous part of this thesis we have provided the background for and state of the art in *designing* self-organizing emergent systems as well as presented several artifacts that can be used for the systematic engineering of effective and in particular efficient self-organizing emergent systems, in this part we provide the background for and state of the art in *operating* these systems as well as present an approach to improve the efficiency of these system at runtime.

As already indicated, there exist several reasons why self-organizing emergent systems (constituting a form of endogenous self-management) are neither able to guarantee optimal solutions to dynamic problems nor to guarantee a required degree of efficiency during operation on their own (cf. *Problem 2*). In Section 5.1, we elaborate on these reasons in more detail. As a consequence, higher level approaches (constituting a form of exogenous self-management) are required that assess the behavior of these systems at runtime and change their behavior or structure when the assessment indicates that a more optimal or efficient solution would be possible. To realize this form of self-adaptation for conventional computer systems, usually a controller is added to the basic system (see also Section 2.2). This concept has its roots in the area of control theory. Section 5.2 hence provides the necessary foundations on control theory as well as different types of control systems. Subsequently, Section 5.3 describes several reference models how to adapt computer systems based on the foundations of control theory. However, with regard to the adaptation of self-organizing emergent systems, the specific characteristics of these systems (see Subsection 2.3.3 and 2.3.6) have to be respected, which complicate the application of these reference models to this special class of computer systems (cf. *Challenges 4–6*). Thus, Section 5.4 surveys existing approaches for adapting self-organizing emergent systems. Finally, Section 5.5 concludes this chapter.

5.1 Runtime Insufficiencies

Self-organizing emergent (multi-agent) systems, as already mentioned, in general are an eligible candidate for solutions to dynamic optimization problems. However, a perfect decentralized coordination with regard to the optimality and efficiency of these solutions is hard to achieve, independent of the efficiency measure and

decentralized coordination approach applied. But in particular industrial settings call for the achievement and maintenance of a certain degree of efficiency by these systems, in order to reduce OPEX. This efficiency should even be maintained in face of the high dynamics, complexity, and unpredictability of the problems to solve.

Approaches that aim to (partly) guarantee this efficiency already in advance, i. e. at design time, are mostly based on extensive simulations prior to the deployment (e. g. [BGP06, GVCO08, SHW08]) but also on elaborated design methodologies [DH05], interactive verification [HRS91], model checking [CGP99], or formal modeling [RS06]. However, these approaches are mainly insufficient, because self-organizing emergent systems are expected to function in open and very dynamic environments with unforeseeable contingencies, i. e. changes may be too complex or too frequent to be completely constrained or predicted in advance. A major issue thereby is the fact that the problems, in more detail the tasks that have to be fulfilled by the system elements respectively agents in a self-organizing manner, usually change dynamically.

Therefore, solving such dynamic problems by self-organizing emergent MASs optimally and efficiently requires as optimal local decisions by the agents as possible with regard to the global solution. However, the agents usually tend to make suboptimal local decisions that result in global inefficiencies of the entire solution. This has the following reasons:

- **Reactiveness of agents:** Because the agents in a self-organizing emergent MAS are usually based on a reactive or hybrid agent architecture (see Subsection 2.4.1) and kept relatively simple, they mainly work in a stimulus-response manner. Instead of any symbolic representation of the world and any abstract reasoning, they make their decisions directly based on the input of their sensors. However, without any perceivable input, the agents may either do nothing or may explore the environment randomly, which both may lead to a suboptimal behavior, depending on the applied solution quality measure.

- **Greediness of agents:** The agents in a self-organizing emergent MAS at every point of time try to make the most optimal decision, however, due to their limited reasoning capability, only with regard to the local level. Unfortunately, this does not necessarily result in an optimal solution to a problem with regard to the global level. In many cases, it might be even better for an agent to make a more suboptimal decision with regard to the local level in order to produce a more optimal solution with regard to the global level.

- **Absence of global knowledge:** In order to make optimal local decisions (with regard to the global level) on its own, an agent would have to be in possession of an abundance of relevant information. This includes information about the environment topology (e. g. networks, machines, customers, etc.), the current and future state of the environment, in particular the problem-relevant tasks, as well as the current and future intended behavior of other agents. This would force an agent to quickly gather real-time information

from a large number of (possibly unknown) entities. Because self-organizing emergent MASs are usually too complex in order to acquire or maintain such a global knowledge, the agents usually only have their own local, subjective view of the entire system, which depends on their perceived situations and performed actions. In particular, the agents neither know about all tasks of a problem to fulfill nor the behavior or position of other agents. Thus, an agent always tends to make suboptimal local decisions, as it might never know, if another agent would be better suited to fulfill a given task.

- **Inability to 'look into future':** In order to create an optimal solution, the agents not only have to be able to find a solution quickly, but also would have to be able to 'look into the future', so that a dynamically appearing task can be assigned to the best agent with respect to the global optimality of the solution, while other tasks are already executed by the agents. But as the agents in a self-organizing emergent MAS are not able to know about future tasks, it is in most cases impossible to solve the entire, dynamically developing problem optimally.

In Subsection 7.3.2 we identify global inefficiencies in environment-mediated self-organizing emergent MAS solutions to PDPs that result from these reasons. Please note that this list of reasons does not claim to be exhaustive. Due to the nature of emergence (see Subsection 2.3.6) it is even not clear, if the identification of all possible reasons resulting in global inefficiencies is possible. However, already this number of reasons clarifies, why the agents in self-organizing emergent MASs are not able to guarantee a certain degree of efficiency on their own, and why higher level control approaches are required, which assess the runtime behavior of these systems and possibly change their behavior or structure to influence the quality of the system output.

5.2 Control Theory Foundations

An area that deals with assessing and influencing the output of dynamic systems in general is control theory – originally an interdisciplinary branch of engineering and mathematics – and in particular control engineering [Bur01, Oga09]. The focus of the latter is on the explicit design and engineering of control systems (see e. g. [Nis07]). In the broadest sense of the term, control systems are what make machines, plants, or systems function as intended (cf. [DFT90]). Thus, a control system can be seen as a device or set of devices to manage, command, direct, or regulate the desired output of a system (cf. [DB07]).

5.2.1 History

The use of control systems has a rather long history (see [Bur01, DB07, Oga09] for more detailed overviews). The first known control system dates back to 270

B.C., in more detail to the water clock of Ktesibios in Greece, which was using a float regulator to keep the water level in a tank at a constant depth. The first control system of modern times, a temperature regulator for a furnace, was developed around 1624 by Cornelis Drebbel. The first automatic control system, i. e. a control system that does not involve a continuous manual control of humans, was invented by James Watt in 1767 by the all-mechanical fly-ball governor. Its purpose was to control the speed of steam engines used in industrial processes. Only around 1868, many years after Watt's invention, Maxwell [Max68] developed a theoretical framework for such governors by means of differential equation analysis relating to the performance of the overall system. With this work the theory of control systems was firmly established. Other significant work in the early stages of control theory were due to Minorsky [Min22], who developed a three-term controller for the steering of ships, Nyquist [Nyq32], who developed a theory for the design of stable repeater amplifiers in the telecommunication area, and Házen [Ház34], who coined the term servomechanism, which has became widely used as a name to describe many types of control systems that imply a master/slave relationship.

Because the theory of automatic control was not much developed until the 1940s, for most systems the design of appropriate control systems was indeed an art. Due to the World War II and the need for automatic aircraft pilots, gun positioning systems, or radar tracking systems, mathematical and analytical models were developed and practiced so that since the 1940s control engineering has been established as an engineering discipline in its own rights. In particular work by Bode [Bod40], who developed the so-called *frequency-response method*, and Evans [Eva48], who developed the so-called *root-locus analysis*, build the core of *classical control theory*.

Since the late 1950s the systems to control have become more and more complex and therewith classical control theory powerless. With the advent of digital computers, thus, *modern control theory* [Bro90] has been developed since about 1960. This theory started with the work by Kalman [KB60a, KB60b] and is based on *time-domain analysis* and the synthesis using state variables. This is why it requires an exact state-space model of the system to be controlled. From then on control engineering has enjoyed tremendous growth and highly sophisticated control systems have been devised and implemented, for instance automatic aircraft landing systems, rocket autopilot systems, or boiler-generator systems. From the 1960s to the 1980s *optimal control* [Kir04] of both deterministic and stochastic systems has been fully investigated. From the 1980s to the 1990s, the developments in modern control theory then were centered around *robust control theory* [Dor87]. It blends the best features of classical and modern techniques, by incorporating classical frequency-domain techniques into modern control systems to improve their stability.

An appealing research field in control theory up to today is *adaptive control* [AW94]. It involves techniques for modifying the models or control laws used by the controller to cope with slowly occurring changes of the uncertain or time-varying parameters of the controlled system (see e. g. [DH02]), such as adaptive dual control [FU00], Model Identification Adaptive Control (MIAC) [SS88], or Model Reference Adaptive Control (MRAC) [AW94]. Adaptive control is different from robust con-

trol in the sense that it does not need a priori information about the bounds on these parameters. In other words, whereas robust control guarantees that if the parameter changes are within given bounds the control laws need do not have to be changed, adaptive control is precisely concerned with the changes of the control laws.

5.2.2 Diagrammatic Representation of Control Systems

For the diagrammatic representation of control systems, in control engineering an application-independent block diagram is used. A block within such a diagram in general may represent an element, a device, a system, etc., whose inner details are not further indicated (black box principle). Each block has inputs (commands) and outputs (controlled variable(s)), whose relationship is characterized by the block. The signal flow through a block is unidirectional.

The most general block diagram of a control system is shown in Figure 5.1. On this abstract level, it consists of a *controlled system*[1] and a *controller*, which influences the operational conditions of the controlled system in a top-down manner. The controlled system consists of everything that is fixed at the start: actuators that generate inputs to the system, sensors that measure certain variables, analog-to-digital and digital-to-analog converters, etc.. The *exogenous inputs* comprise references (the desired output of the controlled system), external disturbances, sensor noises, etc.. The *system outputs* include all the variables that have to be controlled, such as tracking errors between references and system outputs, signals, whose values must be kept between certain limits, etc.. All *sensor outputs* are sent to the controller, whereas all *control inputs* are sent to the controlled system. In general, when one or more output variables of a system need to follow a certain reference over time, the controller manipulates the inputs to a system to obtain the desired effect on the output of the system. The next two subsections will instantiate this abstract control system.

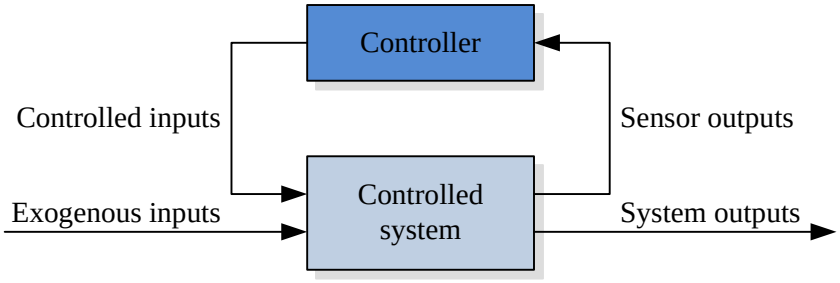


Figure 5.1: Generic control system

[1] In control engineering, the system or object to be controlled is mostly termed *plant*, no matter what it is. But for the purpose of this thesis, we will refer to this system as *controlled system*.

5.2.3 Non-Feedback Control Systems

In general, two major types of control systems can be distinguished: *non-feedback control systems* and *feedback control systems*. In non-feedback control systems, also called open loop control systems, the output of a controlled system has no effect on the controller respectively on the control action (see Figure 5.2[2]). In other words, in non-feedback control systems the system output is neither measured nor fed back for comparison with the reference. Thus, to each reference input there corresponds a fixed operating condition of the controlled system. However, in the presence of external disturbances to the controlled system, an open loop control system will not produce the desired output but the actual output may vary from the desired output in an uncontrolled fashion. Consequently, this type of control system can only be used in practice, if such fluctuations can be tolerated or if the relationship between the input and output of the controlled system is known – i. e. the controller has perfect knowledge of the controlled system, e. g. by exact models – and if there are neither internal nor external disturbances. Open loop control systems are often used for simple basic systems because of its simplicity and low-cost, especially in systems where feedback is not critical (cf. [Oga09]).

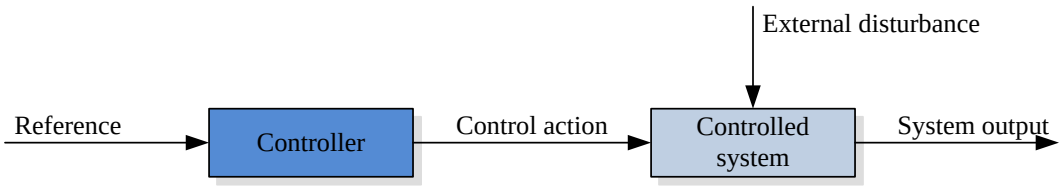


Figure 5.2: Non-feedback control system

An example of an open loop control system is a washing machine. Soaking, washing, and rinsing in the washer operate on a time basis. The washing machine, however, does not measure the system output, that is the cleanliness of the clothes. Another example is an irrigation sprinkler system. The sprinkler system is programmed to turn on at set times. However, it does not measure soil moisture as a form of feedback. Even if it rains, the sprinkler system will activate on schedule.

5.2.4 Feedback Control Systems

To obtain a more accurate control system, in general, it is necessary to feed the system output of the controlled system back to the inputs of the controller. Thus, such feedback control systems, also called closed loop control systems, provide a generic mechanism for self-adaptation. They play an important role in modern systems engineering, because they have the possibility for being adopted to perform their assigned tasks automatically. Thereby, two types of feedback are distinguished:

[2] The non-feedback control system fits in the generic control system depicted in Figure 5.1 by defining the sensor outputs to be always constant.

positive feedback and *negative feedback*. Positive feedback occurs when an initial change in the controlled system is reinforced, which leads toward an amplification of the change. By contrast, negative feedback triggers a response that counteracts a perturbation in the controlled system.

The most elementary feedback control system can be represented as depicted in Figure 5.3 (cf. [DFT90]). This control system is composed of three basic elements: the controlled system, the controller, and a *sensor* to measure the output of the controlled system. In control engineering, actuators are usually lumped in with the controlled system, whereas sensors are considered as elements in their own rights. Obviously, each of these three basic elements has two inputs, one internal to the system and one coming from outside the system (called exogenous input), as well as one single output.

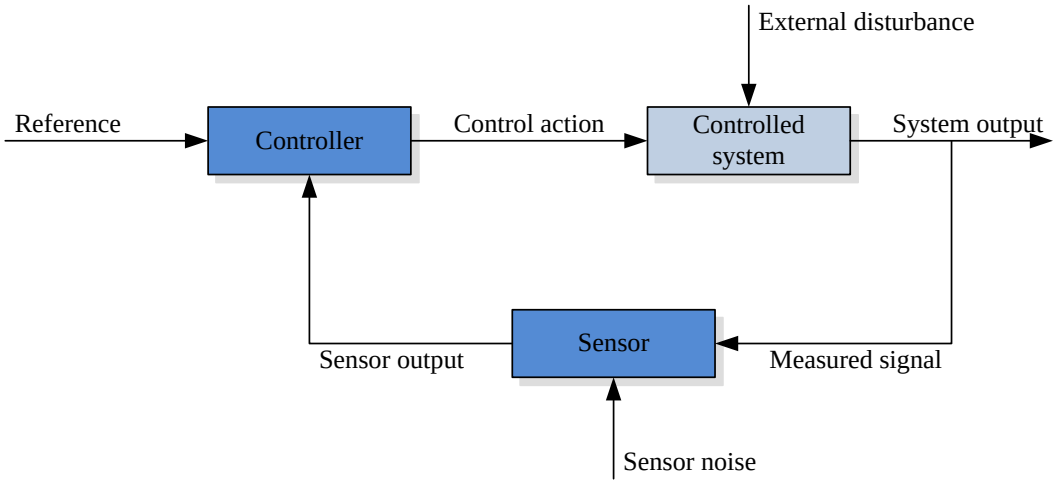


Figure 5.3: Feedback control system

The objective of this elementary feedback control system in general can be summarized by the statement that the system output respectively the measured signal should approximate some specified reference – even in presence of external disturbances, sensor noise, as well as uncertainty in the controlled system. This should be accomplished by a minimum number of control actions, also known as actuating signals. Very often, it makes also sense to describe this objective in terms of the sensor output rather than the system output, since often the only knowledge of the system output is obtained from the sensor output. Therefore, the controller usually comprises a kind of error detector that compares the sensor output with the reference and determines from this difference the control action(s) to be taken (cf. [DFT90]). For further aspects of control systems and the activities of a controller from the control engineering perspective, we refer the reader to [DFT90, FPEN09].

Apart from the engineering branch, such feedback control loops can be found in a variety of other fields. For instance, in economics, the vicious price-wage inflationary cycle, which consists of wages, product costs, and costs of living, forms a kind of

positive feedback control loop as well. As the wages increase, the product costs and therewith the costs of living increase as well. Due to a resulting dissatisfaction of the people, the wages will increase further, resulting in higher production costs, and so on. Another example is for instance present in human societies. There, epidemics in human beings form a kind of negative feedback control loop. In a healthy society, humans have a normal rate of daily contacts. When an epidemic disease affects these humans, infectious contacts actually produce the disease. Due to the isolation of sick people and the medical immunization, however, the infectious contacts are reduced. Another example of negative feedback control can be found in human beings itself. A rise in blood glucose level following the ingestion of food triggers the release of insulin and results in a drop in the glucose level.

In nature, of course, a variety of feedback control examples can be found. For instance, the dropping of pheromones by ants during foraging is a kind of positive feedback control, resulting in ant trails between a food source and a nest (see also Subsection 4.5.1). Even in software engineering feedback control loops can be found. For instance, the design phases in both the extended waterfall model and the unified process are based on feedback from e. g. the test phases.

5.3 Reference Models for Self-Adaptive Systems

As explained in the last section, control theory and control engineering provides excellent descriptions of closed systems whose components and desired outputs are known. Consequently, the adoption of the concepts and approaches, in particular feedback control loops, is considered to be advantageous for the area of computer science as well, more specifically for the engineering of self-adaptive respectively (exogenous) self-managing systems (see e. g. [KBE99, HDPT04, DHP+05, CGI+09]).

From the perspective of software engineering, feedback control loops typically involve four key activities (cf. [BDG+09]): collect, analyze, decide, and act (see Figure 5.4). These activities are consequently arranged in a closed loop, also called feedback cycle. The main focus of the depicted feedback control loop is rather on the activities that realize the feedback cycle than on the properties of the control and data flow around the cycle. Please note that this generic model of a feedback control loop represents a refinement of the AI community's *sense-plan-act* approach of the early 1980s to control autonomous mobile robots (cf. [RN02]).

The feedback cycle starts with the collection of relevant data from environmental sensors and other sources that reflect the current state and the context of the controlled system. Essential factors for this activity are the sample rate and the reliability of the sensor data with regard to the sensor noise. Furthermore, the data format used by the sensors influences this activity as well as the completeness of the information provided by the sensors for system identification.

Subsequently, the controller analyzes the collected data. Therefore the collected data first is cleaned, filtered, pruned, and, finally, stored for future reference to portray an accurate model of past and current states. Then, the diagnosis analyzes

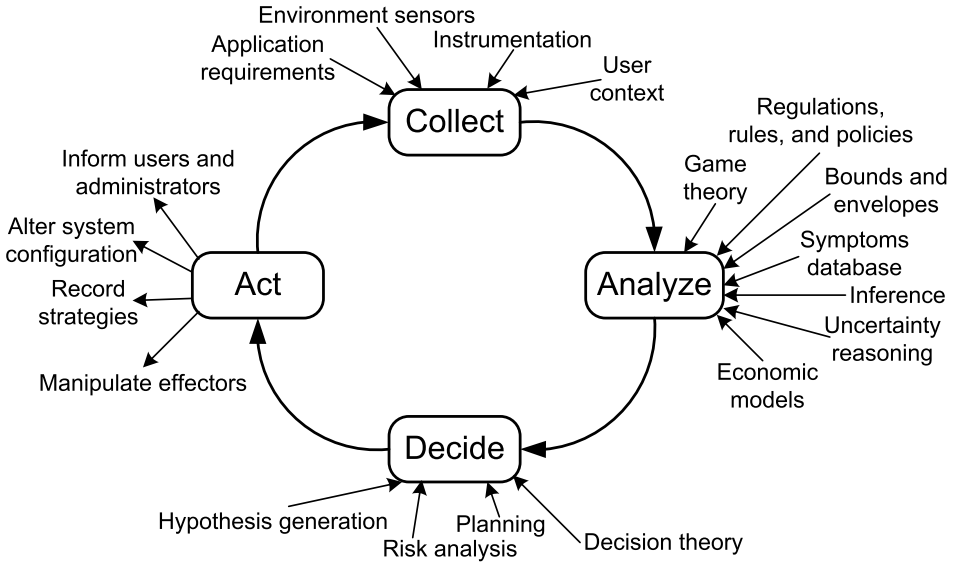


Figure 5.4: Activities of a feedback control loop [DDF+06]

the stored data to infer trends and identify symptoms, e. g. by using models, theories, equations, or rules. Essential factors for this activity are the amount of past states that may be needed in the future, the archiving of data for validation and verification, the adequateness and faithfulness of the model compared with the real world, and the stability of the model over time.

Third, the future must be predicted, e. g. by off-line simulation, utility/goal functions, or system identification, and a decision must be made about how to adapt the system in order to reach a desired system response or state. Approaches such as risk analysis help in choosing among various alternatives.

Finally, to implement the made decision, i. e. to adapt the executing system and possibly its context, the controller must act via available actuators or effectors. Essential factors for this activity are the time when the adaptation can be safely performed. The impact of the decisions respectively adaptations can then be collected again to inform the next feedback cycle.

Instances of this generic feedback control loop model can be found in different self-adaptive solution approaches to various problems, e. g. selected resource management problems such as task scheduling in real-time [CEBÅ02, LSST02, HSM+07] and embedded systems [SKS+08], bandwidth allocation and QoS adaptation in web servers and applications [ASB02, DHP+06], load balancing and throughput regulation in email and file servers [LAW02, PGH+02], network flow control [Mas99, Sri00], and power management [LHH+02, SBG04].

Although control theory provides well-established mathematical models, tools, and techniques to analyze system performance, stability, sensitivity, or correctness – which in principle can be used and are used to adapt the parameters of a controlled

computer system – self-adaptive systems also have to be able to make structural changes in the controlled computer system in forms of compositional adaptation (cf. Section 2.2). This may require rather different forms of feedback loops, which go beyond classical, robust, or adaptive control theory. As a consequence, very often, multiple, intertwined feedback loops will be involved in a practical self-adaptive system. Good engineering practice, however, calls for reducing multiple control loops to a single one or making control loops independent of each other [BDG+09]. Another typical scheme, which originates from control engineering, is to organize multiple feedback loops in the form of a hierarchy. Then, due to employed different time periods, unexpected interferences between the hierarchy levels might be excluded (cf. [HGB10]).

Due to the occurrences of feedback loops in various fields mentioned in the last section, especially in nature, there already exist a couple of approaches for self-adaptive systems, which implicitly exhibit one or more feedback loops due to their biological inspiration (see e. g. [AAC+00, Brü00, CR03]). However, the feedback behavior of a self-adaptive system, which is realized with feedback loops, is a crucial feature for the proper engineering of these systems. The properties of the feedback loops affect the systems' design, architecture, and capabilities. Thus, the self-adaptive systems community advocates to elevate feedback loops to a first-class entity in the modeling, design, implementation, validation, and operation of self-adaptive systems (cf [CGI+09]). Similarly, Brun et al. [BDG+09] advocate to make feedback loops as well as the properties of feedback loops explicit, whereas Müller et al. [MS08] advocate in more detail to make feedback loops explicit in design and analysis and either explicit or clearly traceable in implementation. Garlan et al. [GCS03, CGS05] additionally advocate to make self-adaptation external, as opposed to internal or hard-wired, in order to separate the concerns of system functionality from the concerns of self-adaptation (see also Section 2.2). Consequently, in the following we will only consider influential reference models and architectures for self-adaptive systems, which incorporate feedback loops in an explicit manner.

The general foundation for self-adaptive systems in the sense of exogenous self-management is provided by Shaw [Sha95]. She introduced a new software organization paradigm based on process control loops with an architecture that is dominated by feedback loops and their analysis.

Oreizy et al. [OGT+99] propose an architecture-based approach to self-adaptive systems. The idea behind this approach is based on two simultaneous processes: *system evolution*, i. e. the consistent application of change over time, and *system adaptation*, which is responsible for detecting changing circumstances and planning and deploying responsive modifications. This approach is the first that highlights the possibility of an explicit system representation in the form of software architectures, which can be used by the adaptation mechanisms. In such approaches, see e. g. [CGS+02a, CGS+02b, KLS+03] in the following years, software architectures are used to describe, understand, and specify the software artifacts, their interrelationships, as well as principles and guidelines governing their design and evolution over time. Thus, software architectures, on the one side, provide a global perspec-

tive on the systems, and on the other side explicitly express constraints and hence help to ensure the validity of adaptation. Furthermore, they are separated from the applications, which supports the required separation of concerns for self-adaptation.

Garlan et al. [GS02, GCH+04] similarly focus on the idea of an abstract system representation by introducing an explicit layer for model management. This layer can then be used by high-level adaptation mechanisms. Dashofy et al. [DvdHT02] show that the idea of an abstract system representation can be expanded to an integrated approach, in which an architectural description language (ADL) offers explicit support for self-adaptation.

Kramer and Magee [KM07] describe a reference architecture for self-adaptive systems by proposing a component-based architectural approach for self-management (see Figure 5.5). In the style of robot architectures [Gat98], this high-level layered software architecture comprises three layers: component control as the bottom layer, change management as the middle layer, and goal management as the top layer. The bottom layer consists of the set of interconnected components that accomplish the application function of the system. It must of course include facilities to report the current status of components to higher layers and also include the capability to support component creation, deletion, and interconnection. The change management layer reacts quickly to changes in state reported from the bottom layer and executes pre-specified plans, which are activated in response to the state change, that select new control behaviors and set new operating parameters for existing control layer behaviors. This layer can introduce new components, recreate failed components, change component interconnections, and change component operating parameters. If a situation is reported for which a plan does not exist then this layer must invoke the services of the goal management layer. This upper layer consists of time consuming computations such as planning, which takes the current state and a specification of a high-level goal and attempts to produce a plan to achieve that goal. It produces change management plans in response to requests from the change management layer and in response to the introduction of new goals by users. The change management layer is then responsible for effecting changes to the underlying component architecture in response to new states reported by that layer or in response to new objectives required of the system.

One of the most well-known reference models for self-adaptive respectively self-managing systems is described in the IBM architectural blueprint [IBM06] (see Subsection 2.5.1). There, the Autonomic Manager (AM) realizes an autonomic (feedback) control loop (see Figure 5.6) to control multiple managed elements, such as processors, databases, or servers. The AM thereby features the feedback control loop as a central architectural component.

Similar to the generic model of a feedback control loop depicted in Figure 5.4, the reference model provided by IBM dissects the autonomic control loop into four functions sharing common knowledge. The *monitor* function provides the mechanisms that filter, aggregate, and report details collected from a managed resource. These details are collected through a sensor provided by the manageability endpoint allocated to the managed resource. Based on these details, the *analyze* function pro-

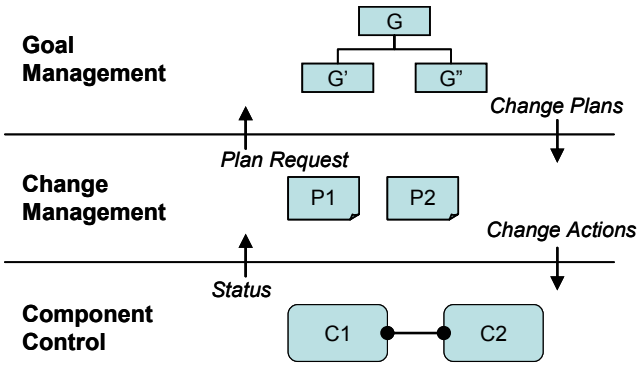


Figure 5.5: Three layer architecture model for exogenous self-management [KM07]

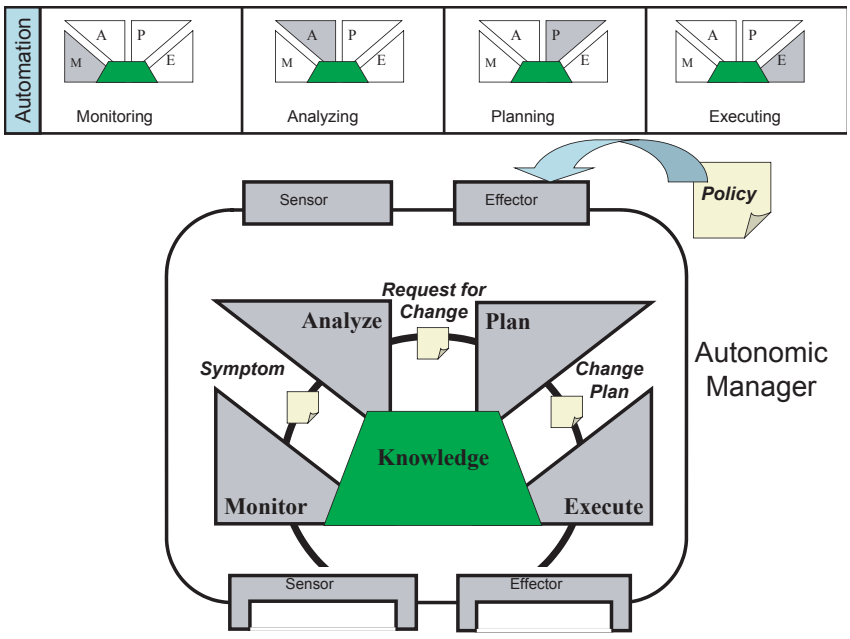


Figure 5.6: Reference model for autonomic control loops [IBM06]

vides the mechanisms that correlate and model complex situations, e. g. time-series forecasting and queuing models. These mechanisms allow the AM to learn about the IT environment and help to predict future situations. The *plan* function provides the mechanisms that construct the actions needed to achieve goals and objectives. Therefore, the planning mechanism uses policy information provided from the outside by a human administrator to guide its work. The *execute* function provides the mechanisms that control the execution of a plan by communicating instructions, parameter changes, or updates through a effector provided by the manageability endpoint to the managed resource. Note, in the same manner as manageability endpoints, AMs provide sensor and effector interfaces for other autonomic or manual managers and other components in the distributed infrastructure to use.

The more of the four functions of a control are automated, the more self-managing capability is provided, although a human administrator may decide to delegate only portions of the potential automated functions to the AM, as illustrated in 5.6. Due to the functions, this control loop is sometimes also called MAPE-K loop (**M**onitor, **A**nalyze, **P**lan, **E**xecute, **K**nowledge). The reference model of this control loop is being used more and more to communicate the architectural aspects of AC systems. Likewise, it is a clear way to identify and classify much of the work that is being carried out in the field (see [Ste05, SPTU05, HM08] for more detailed information). In literature, various instantiations of this generic reference model can be found in different application areas, e. g. power management [KHY08] or service centers [MB07], to mention just a few.

5.4 Adapting Self-Organizing Emergent Systems

Despite the achievements of the reference models described in the previous section to realize self-adaptive systems, their application for the adaptation of self-organizing emergent (multi-agent) systems is complicated. In contrast to conventional systems, the latter class of systems exhibits specific characteristics (see Subsection 2.3.3 and Subsection 2.3.6) that have to be respected for their adaptation, more specifically, when using principles of self-adaptation in order to compensate for the runtime insufficiencies of self-organizing emergent systems listed in Section 5.1. These characteristics result in the following general constraints for the adaptation of self-organizing emergent systems (cf. *Challenges 4 – 6*):

- **Low observability and poor controllability:** Controllers for conventional computer systems, for instance the AM proposed by IBM (see previous section), depend fundamentally on the assumption that they can be equipped with the corresponding capabilities to observe and control the activities of all subsystems or lower level system elements at any time and, in case of mobile systems, any place. This becomes feasible due to the inherently high observability and good controllability of conventional computer systems. By contrast, this assumption in general is not applicable to self-organizing emergent systems. In these systems, which usually consist of a multitude of possibly mobile

elements, communication might be very costly (in terms of required communication resources), locally forbidden (e. g. in certain application domains such as hospitals), globally restricted (e. g. when using indirect communication), structurally infeasible (e. g. in ultra large scale systems), or only temporally possible (e. g. in space missions). Thus, the information flow between an element of the controlled system and a controller (as well as vice versa) might not be possible at any time or any place. The information flow is rather only possible at distinct or random points in time or places. Furthermore, because in these systems every element in general is exposed to incomplete information only, gathering all required information to control or adapt the entire, possibly large scale, system in real-time is a complex endeavor.

- **Capability for self-organization and emergence:** The beneficial properties of self-organizing emergent systems, such as their scalability, robustness, flexibility, and adaptivity, are in contrast to conventional computer systems a result of their capabilities to self-organize their actions and interactions as well as to build emergent properties. As a consequence, any controller for these systems is not supposed to limit these capabilities. This implicates that any controller may neither become a processing bottleneck nor a single point of failure, but has to preserve the basic behavior of the controlled system. In particular, a controller may not act as an omnipotent supervisor that strictly guides and determines all actions and/or interactions in the controlled system. In particular, the basic behavior of the controlled system has to be preserved and all problem solving decisions still have to be made locally by the system elements themselves.

- **Openness and autonomy:** Self-organizing emergent systems are intended to operate autonomously in previously (i. e. at design time) unknown or unexpected situations. Furthermore, the number of elements in these systems are subject to a continuous change, for instance due to unexpected breakdowns of some elements or new elements joining the system. Similarly, any controller for these systems has to be able to cope with such open situations as well. This implicates that the assessment of the controlled system's behavior has to consider the actual (current and past) situations, while the behavior or structure of the system has to be adapted depending on these possibly unforeseen situations. Furthermore, in order to achieve the overall objective to achieve a high degree of autonomy, not only the controlled system but also the adaptation process realized by a controller has to exhibit a high degree of autonomy as well. Furthermore, if the controller crashes, the system elements still have to function properly, i. e. they have to solve the problem at least as good as without any adaptation.

Please note that these constraints apply to self-organizing emergent system in general, even though there may be systems defined for which only parts of them apply. For instance, if an engineer is able to guarantee a reliable communication

between the system elements and a controller at any time and place, and the system elements are equipped with sufficient communication resources, the first constraint will be weakened to some extent. Likewise, if an engineer assumes the number of elements in a self-organizing emergent system to be fixed in all possible situations, or all possible situations are known already at design time, the last constraint will be weakened to some extend.

Nonetheless, in general these constraints have a strong impact on each of the four key activities of a feedback control loop that has to be realized by an appropriate controller (see Figure 5.4). Because the *analyze* and *decide* activities depend on the *collect* and *act* activities, the description of the requirements starts with the latter:

- **Collect:** Due to the low observability and the openness of the controlled system, a controller in general is not able to observe every (inter)action of (possibly unknown) system elements at the time of occurrence, if ever. Due to the openness of the controlled system a controller is rather dependent on the information provided by the system elements to the controller. In other words, instead of a pull strategy to collect information, a push strategy is more appropriate. Because the information flow between the system elements and a controller in any case can only happen when communication is possible (with regard to the communication constraints) or the system is small enough, a controller usually cannot collect enough information to determine the controlled system's *current* state and context. By contrast, a controller is rather only able to collect enough information to determine the controlled system's *past* state(s) and contexts. However, even this amount of information might not be complete, e. g. if some system elements were not able to provide their information (yet).

- **Act:** Due to the low observability (and the associated complexity to determine the current state of a controlled system) and poor controllability of the controlled system, a controller cannot send specific control actions to dedicated system elements as soon as these actions are provided by the *decide* activity. By contrast, a controller has to wait until the communication constraints, if any, allow for the transfer of this information to the specific elements. Thus, a controller in general is not able to adapt or influence (control) the behavior, intention, or upcoming action of any system element yielding immediate effects.

- **Analyze:** Because the controlled system in general cannot be influenced or controlled by a controller yielding immediate effects, a controller in general is forced to adapt, i. e. to change, the local behavior of system elements in order to yield future effects. In other words, the local behavior of the system elements has to be adapted in a way such that the elements are able to autonomously behave more optimally in future situations themselves, because in these future situations no immediate control actions by a controller will be

available. Therefore, in the *analyze* activity, a controller is required to autonomously predict possible future situations, in which the controlled system will exhibit certain runtime inefficiencies. For this prediction some kind of online learning is required, because the controller has to take into account the actual problem, which the controlled system is intended to solve, but which is in general previously unknown to the controller.

- **Decide:** Instead of determining control actions that immediately influence the system output, the *decide* activity rather has to determine appropriate adaptations of the local behaviors of the system elements for the future situations predicted by the *analyze* activity. Due to the openness of the controlled system and the dynamics of the problem to solve, a controller however has to consider the fact that any predicted situation in principle may occur, but not necessarily must occur. This fact makes very rigid adaptations by the controller useless and underpins the constraint that each problem solving decision still has to be made locally by the system elements themselves in order to preserve the basic self-organizing and emergent capabilities. Furthermore, it is important to note that a controller can only adapt the system elements of the controlled system but in general not the environment of the controlled system, which is usually not controllable.

Apart of several approaches to adapt (self-organizing emergent) MASs, which are restricted to certain problem or application domains (see e. g. later Subsection 7.1.3 for the PDP domain), up to today, there only exist a few general approaches to assess the behavior of a self-organizing emergent system and – if necessary – to provide regulatory feedback to control and adapt its dynamics. However, all of these approaches make certain assumptions to the controlled system and the controllers' capabilities, which do not respect all of the above constraints and consequences. Again, we only consider approaches that make the feedback control loop explicit.

5.4.1 Observer/Controller Architecture

In the area of OC (see Subsection 2.5.2), a very general approach is presented by the generic observer/controller (O/C) architecture [RMB+06] (see Figure 5.7).

Approach

On an abstract level, this instantiation of a feedback control system consists of three components: (1) a self-organizing emergent controlled system, termed *system under observation/control* (SuOC), (2) an observer, which is in charge of identifying and characterizing the nature of an (emergent) phenomenon representing the current state of the SuOC as well as predicting the future state of the SuOC, and (3) a controller, which takes appropriate actions to influence the SuOC. On the one side, this generic architecture can be mapped to the reference model of an AM provided by IBM (see Figure 5.6), that is the *observer* component encapsulates the *monitor* and

analyze functions whereas the *controller* component encapsulates the *plan* and *execute* functions. On the other side, it can be mapped to the generic feedback control loop model (see Figure 5.4) as well, that is the *observer* component realizes the *collect* and *analyze* activities whereas the *controller* component realizes the *decide* and *act* activities. In contrast to these two abstract models, the generic O/C architecture however recommends more specifically, which functions should be implemented in order to realize a feedback control loop for adapting self-organizing emergent systems. In particular, it emphasizes the necessity of online-learning (to quickly react on changing situations) and offline-learning (to identify optimal parameter sets).

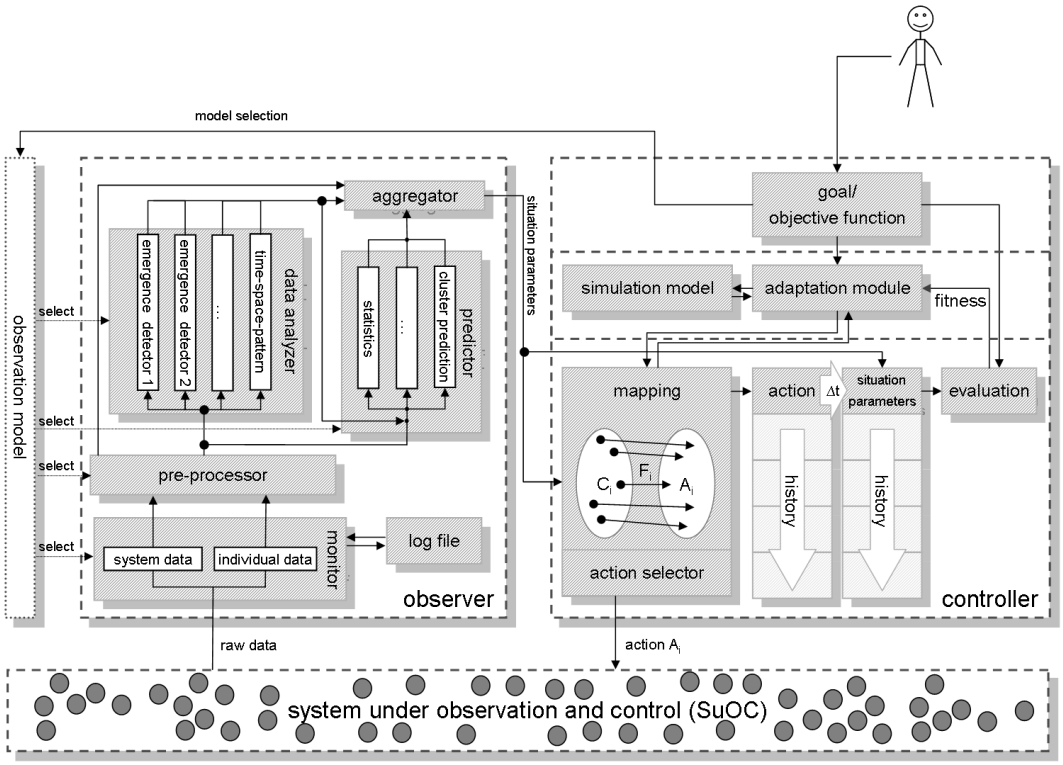


Figure 5.7: Generic observer/controller architecture [BMMS+06]

In more detail, the authors provide the following recommendations for the realization: the observer in an O/C architecture performs an aggregation of available information on the SuOC in form of indicators to give a global description of the *current* state and the dynamics of the controlled system. The observer is guided by an observation model, which is responsible for selecting observable attributes, selecting appropriate analysis tools with regard to the purpose given by the controller, and selecting appropriate *prediction* methods. The observation process then consists of several steps: the monitoring step samples raw data coming from the SuOC to generate time series, which reflect the current state of the SuOC as well as its history. All measured data is stored in a log file for every loop of observing/con-

trolling the SuOC. From the raw data, some derived attributes can be computed by a pre-processing step. The pre-processing of the raw data also includes a selection of relevant data, which is required to compute aggregated system-wide parameters. Based on this data, a data analyzer applies a set of detectors, e. g. for the computation of clustering, the detection of emergence, or some mathematical and statistical values. At the end of this step a system-wide description of the current state is provided. The predictor process uses this description, e. g. by own methods or some methods of the data analyzer combined with prediction methods taken from technical analysis, to *predict the future behavior of the SuOC* in order to reduce the reaction time of the controller (online learning). All results are then handed on to an aggregator, which delivers a set of filtered current and previous values to the controller. This constitutes an abstract description of the current state and the dynamics of the SuOC.

According to the recommendations of the authors, the purpose of the controller component can be manifold: (1) to influence the SuOC such that a desired emergent behavior appears, (2) to disrupt an undesired emergent behavior by the SuOC as quickly and efficiently as possible, and (3) to construct the SuOC in a way such that no undesired emergent behavior can develop. To guide the self-organization process between the elements, but to interfere only when necessary, the controller in principle may influence the basic SuOC by (1) influencing the local decision rules of the SuOC elements, (2) the system structure including e. g. the communication between the SuOC elements or the number of elements, or (3) the environment, which will indirectly influence the system by changing the data observed by the SuOC elements through their local sensors. For the influence, three feedback control loops are distinguished: for a quick reaction in real-time, the first loop simply applies the most appropriate action out of a (previously created) set of actions for the current situation and forwards the chosen action to the SuOC. The second loop proceeds concurrently to the first one and keeps track of history data. It measures the situation at a specific time later in order to evaluate the impact and success of a control action, which are updated in the situation-action mapping. The third loop generates completely new rules and modifies existing rules, e. g. by evolutionary algorithms, using a simulation module to predict the success of control actions before applying them to the real SuOC (offline learning).

Obviously, the description of the generic O/C architecture is kept very general as it only provides recommendations for functions that have to be realized in order to implement a feedback control loop on top of self-organizing emergent systems. Thus, the observation model, the observer component, and the controller component have to be customized for each individual application domain. In the following, we hence investigate various existing customizations of the generic O/C architecture.

Instantiations

In [MRB+07], a customized instantiation of the generic O/C architecture is used to control the self-organizing behavior of robot swarms, in more detail of chickens

(which are considered to be autonomous robots or agents with simple rules and local goals). Because densely packed chickens in cages exhibit a collective cannibalistic behavior, when a chicken is wounded, other chickens chase this chicken and pick on it until it dies. In order to minimize the number of killed chickens, the customized O/C – in a simulation – permanently collects data from the SuOC at a fixed sampling rate, in order to determine based on the current state in real-time, if a problematic clustering of chickens starts to emerge at a certain place. If an emerging cluster is predicted (the predication is based on Shannon's information theory), a noise signal with fixed intensity and fixed duration is applied by the environment around the computed cluster centroid to frighten the chickens and disperse the cluster. Thus, in this instantiation, the SuOC is controlled indirectly by the centralized O/C in order to disrupt an undesired emergent behavior.

In [BMMS+06], and more specifically in [PRT+08], the generic O/C architecture is customized for the control of an urban traffic network. In this scenario, the goal is to minimize the average delay per vehicle passing an intersection. Therefore, every traffic light controller for an intersection in the network is equipped with its own O/C for parameter adaptation, i. e. every O/C unit is only in charge of adapting one single system element. While in this decentralized approach every observer component is responsible for analyzing the current traffic situation at its intersection (and possibly predicting trends in the traffic development), every controller decides when and how to change the parameter set for the traffic light controller. Thereby, the controller maps traffic situations to parameter sets and keeps track of how well each parameter set performs. Alternative parameter sets are generated by offline learning, using an evolutionary algorithm and a simulation of the traffic network as testbed, adjusted to the traffic situation in question. To learn good control strategies for these situations, the controller makes use of a learning classifier system. To manage the complexity, the input space in terms of different traffic situations is partitioned, which limits the openness of the system. Thus, in this instantiation, the local decision rules of the elements of the SuOC are influenced directly to optimize the solution locally, even though the basic behavior is not preserved but changed completely. Therefore, all problem-solving decisions are made by each O/C unit. However, no global optimization of the entire urban traffic network takes place, as a coordination between all O/C units is missing.

For an intersection without any traffic lights, in [CHMS08] a fully decentralized instantiation and a centralized instantiation of the generic O/C architecture are presented and compared. In this scenario, the goal of each vehicle similarly is to cross over an intersection as soon as possible, which consequently produces competition situations, since two (or more) vehicle may want to occupy the same position at the same time. The O/C architecture thus is used to create a collaborative group behavior in order to avoid traffic jams. In the decentralized instantiation, each vehicle is endowed with its own O/C unit. Based on local rules and the observation of neighboring vehicles, every O/C unit sends either a 'stop' or a 'go' signal to its corresponding vehicle. In the centralized instantiation, a central O/C unit can interact with all vehicles in an intersection in the same way, having unlimited observation

and control over all vehicles. The authors demonstrate experimentally that their centralized instantiation does not scale with an increasing number of conflict situations between the vehicles. Although both O/C instantiations optimize the entire solution, they do not consider the autonomy of the vehicles nor adapt their local behavior but control every action at every time.

In [RRS08], an instantiation of the generic O/C architecture is presented in order to control and prevent bunching of a set of elevators in a large building. The latter is an emergent misbehavior, in which different elevators synchronize and behave like a huge, single elevator with the capacity equal to the sum of the individual elevators. To disrupt this undesired emergent behavior, the set of elevators is augmented by a single, centralized O/C unit. Thereby, the observer component monitors the current state of the set of elevators in real-time. If a measured situation exceeds a certain threshold, the controller component will then indirectly influence the set of elevators by either hiding all calls for all or even a subset of elevators or by hiding specific calls only from specific elevators. This influence accelerates certain elevators and thus prevents bunching. In this instantiation, the SuOC again is controlled indirectly by the centralized O/C unit, able to observe the entire SuOC at any time and place, as well as able to control the environment.

In [BAU+10], an instantiation of the generic O/C architecture is presented for controlling in-house electrical appliances with respect to energy efficiency. In this scenario, each considered household appliance is equipped with a local O/C unit. The observer component of each unit monitors the current state of its appliance and based on the measured data generates a specific energy demand set. This data is then communicated to a central management component, also equipped with a O/C unit. Obviously, all O/C units together form a kind of hierarchy. The central observer component of this O/C unit generates a global demand set prediction for the entire smart-home. Based on that, the central controller component then decides to individually re-schedule the demand sets of each appliance, if possible, based on the demand prediction from the central observer component and the received load prediction of the energy provider. The aim is to re-schedule demand sets to time slots with a low overall energy demand, thus balancing energy supply and demand. After this re-scheduling, a set of rules is generated and sent to the controller component of each appliance's local O/C unit. These rules contain instructions for an appliance, in which time slot it should start or break its operation. The local controller component therefore contains a simple set of static rules to interact with the appliance, that is to turn it on or off.

5.4.2 Management-By-Exception

In the area of DAI respectively MASs (see Subsection 2.4), another realization of a feedback control loop is presented by the management-by-exception (MBE) approach [SLT08].

Approach

The idea behind management by exception in economics is that the management devotes its time to investigating only those situations in which actual results differ significantly from planned results. By transferring this idea to the control of MASs, it should allow the controlled MAS as much flexibility as possible by keeping the time of central control as short as possible. The MBE approach is intended to tackle the performance of MASs for dynamic optimization problems.

Instantiation

Up to today, there only exists one instantiation of the MBE approach, which is applied to job shop scheduling problems [Pin08]. By monitoring the finished jobs in fixed intervals, the manager agent, which realizes the controller in this instantiation, is able to measure a violation of the mean flow time of a job, i. e. an exception. In these cases, the manager agent takes over central control and orders all shop agents (machines) to work with a fixed dispatching strategy that is known to perform well. Due to this strategy, all agents are ordered not to optimize their local goals but to work cooperatively optimizing the overall system's performance. As soon as the mean flow time reaches an acceptable range again, the central control is released and all entities can perform their local optimization based on their regular behavior again.

5.5 Conclusion

In this chapter we have provided the background for and state of the art in operating self-organizing emergent systems. We have analyzed reasons, why these systems are neither able to guarantee optimal solutions to dynamic problems nor to guarantee a required degree of efficiency during operation on their own. Furthermore, we have explained several constraints (*Challenges 4–6*) and requirements for the assessment of the behavior of these systems at runtime and the adaptation of their behavior or structure when the assessment indicates that a more optimal or efficient solution would be possible. At the end, we have described existing, general approaches for the adaptation of self-organizing emergent systems.

Table 5.1 compares these approaches respectively instantiations based on the aforementioned constraints and requirements. Apparently, all approaches take a high observability of the controlled system for granted, i. e. the respective controllers are able to observe any element of the controlled system at any time and any place in the environment, which contradicts the general constraints. Thus, all controllers are able to determine the *current* state of the entire system in real time. In regard to the controllability, all but one approach likewise takes a good controllability of the controlled system for granted, i. e. the respective controllers are able to influence any upcoming action or perception of any element of the controlled system at any time and any place in the environment, which contradicts the general constraints

again. Only in [MRB+07] the authors regard a limited controllability of the system elements, which is why they take a good controllability of the system environment for granted. Consequently, almost all approaches aim at influencing the current state of the controlled system achieving immediate effects. Only in [BAU+10], the influence of future states of the controlled system is considered.

	O/C [MRB+07]	O/C [PRT+08]	O/C [CHMS08]	O/C [RRS08]	O/C [BAU+10]	MBE [SLT08]	EIA
Regard for low observability							+
Regard for poor controllability	(+)						+
Adaptation of local element behavior		+				+	+
Preservation of basic behavior	+			+			+
Focus on system performance		(+)	+		+	+	+
Consideration of openness	+		+	(+)	+	+	+
Consideration of autonomy	+	(+)		+	+	+	+
Incorporation of online learning		+			+		+
Incorporation of offline learning		+					

Table 5.1: Comparison of approaches for adapting self-organizing emergent systems

The way how the controlled system is influenced varies. This is either accomplished indirectly by controlling the environment [MRB+07, RRS08], by using a separate controller for each system element that strictly prescribes every action [CHMS08, BAU+10], or by actually adapting the local rule set of the system elements [PRT+08, SLT08]. However, these adaptations then are very rigid and consist either of changing the entire parameter set [PRT+08] or the entire rule set [SLT08].

In [PRT+08, CHMS08, BAU+10] the problem-solving decisions are made on the controller level, whereas only in [MRB+07, RRS08, SLT08] they are made on the element level. In the latter three approaches, however, only in [MRB+07] and [RRS08] the basic self-organizing and emergent behavior is preserved, while in [SLT08] a completely different behavior is prescribed by the controller, which is no more self-organizing and emergent. Please note that the preservation of the basic behavior in [MRB+07] and [RRS08] only results from the fact that the local element behavior is not changed by these approaches in order to influence the system behavior, but the system behavior is influenced indirectly (see above).

Whereas the focus of two instantiations is to disrupt an emergent misbehavior [MRB+07, RRS08] in the controlled systems, most approaches focus on the performance of the latter. However, the approach in [PRT+08] is only able to achieve a locally optimized solution, whereas the decentralized approach in [CHMS08] disre-

gards the global optimality of the solution entirely. This results from the decentralized instantiations of the O/C architecture, whereas the other approaches are realized centrally or hierarchically.

The openness of self-organizing emergent systems with regard to unknown situations and a changing number of system elements to be controlled by one controller is considered by most approaches [MRB+07, CHMS08, BAU+10, SLT08]. Only in [RRS08] a fixed number of system elements is supposed, while [PRT+08] additionally limits the number of situations the system can be in.

If the respective controller crashes, most approaches will be able to continue working properly, i. e. to achieve a performance that is as least as good as without any adaptation. Apart from the two approaches intended for the disruption of undesired emergent behavior [MRB+07, RRS08], where the chance of an undesired emergent behavior becomes very likely as soon as the controller is crashed, in [BAU+10] a crash of any O/C unit will only reduce the system performance to some extent, at the worst case to a level similar to no adaptation. If in [SLT08] the controller crashes in the phase of central control, the agents might not be able to switch back to a self-organizing emergent behavior. However, they will continue to work with a known good performance, although they will lack the beneficial properties of self-organization and emergence. If a O/C unit crashes in [PRT+08], the respective traffic light controller will continue to work as well, however, maybe with a significantly reduced performance, in case that the actual parameter set does not fit the current traffic situation. By contrast, in [CHMS08] the controlled system will stop working, if the controller crashes.

Learning is generally neglected in most approaches. Whereas online learning is applied in [PRT+08] respectively [BAU+10] to learn different traffic situations respectively demand patterns, offline learning is only applied by [PRT+08] to identify optimal parameter sets.

Because none of the existing approaches is able to consider all general constraints for the adaptation of self-organizing emergent systems (see Section 5.4), in the next chapter we present the Efficiency Improvement Advisor (EIA) approach that helps to overcome these drawbacks (see Table 5.1). This approach does not take a high observability and good controllability of the controlled system for granted, but assumes that the EIA, as a form of controller, due to various communication constraints is only able to observe a subset of elements of the controlled system (which may be all elements as well) only at distinct points in time and distinct places in the environment. Consequently, the EIA can only determine past states of the system, but not the entire current state. Therefore, the EIA is able to adapt the local behavior of the system elements for future situations. In contrast to existing approaches, the EIA approach adapts the local behavior of the system elements by only giving advice, how to behave more optimally in future situations. If these situations do not occur in future, all problem-solving decisions are still made by the system elements with their regular local behavior, which is preserved. As the name implies, the primary focus of the EIA approach is to improve the efficiency of the global solution, even though other foci might be conceivable as well. The EIA approach considers the

openness of self-organizing emergent systems with regard to unknown situations as well as a changing number of system elements. If the EIA crashes, the controlled system will continue to work at least as good as without any adaptation. The EIA furthermore applies online learning to provide a high degree of autonomy, whereas offline learning is reserved for future work (see Section 9.3).

Chapter 6

Efficiency Improvement Advisor

As described in the last chapter, the absence of global knowledge, the inability to 'look into the future', as well as the reactiveness and greediness of the agents in self-organizing emergent MASs do not facilitate guarantees on the runtime efficiency of these systems when solving dynamic problems (cf. *Problem 2*). The low observability and poor controllability of these system, their basic self-organizing and emergent behavior, as well as their openness and autonomy, however (cf. *Challenges 4–6*), complicate the integration of a closed feedback control loop on top of these system to provide regulatory feedback to control their dynamics. However, because in particular industrial settings call for the achievement and maintenance of a certain degree of efficiency by these systems in order to reduce OPEX, in this chapter we formally present an approach that autonomously adapts the local behavior of agents in self-organizing emergent MASs in order to improve the efficiency of the global solution in certain situations.

Therefore, Section 6.1 explains the terminology used for this approach, i. e. basic definitions about the general problem setting the agents in a self-organizing emergent MAS are supposed to solve. Based on these definitions, Section 6.2 presents the generic model of an EIA, which is independent of any application domain, agent model, and coordination model. Because thus the model can be realized and customized in different ways, Section 6.3 mentions some aspects of the generic model that are of worth to think about before its realization. Finally, Section 6.4 mentions related work to the EIA approach, before Section 6.5 concludes this chapter.

6.1 Terminology

In order to improve the efficiency of self-organizing emergent MASs at runtime, we assume that these systems were designed to solve certain *problems*, more specifically *problem classes*, which share a common representation and require certain capabilities from the agents. A *problem instance* is then one specific instantiation of a problem. The general structure of the problems we are interested in consists of tasks out of a set of tasks T, which are given to the set of agents A of a self-organizing emergent MAS within a given time interval $Time$. A task denotes the smallest unit of work within a problem.

Definition 6.1 (Task)
A task $ta \in T$ is defined as a triple

$$ta = (Prop_{ta}, t_{ta}^{start}, t_{ta}^{end})$$

where

- $Prop_{ta}$ is a set of properties of ta
- $t_{ta}^{start} \in Time$ is the point in time from that on ta may be executed
- $t_{ta}^{end} \in Time$ is the point in time at which ta must be fulfilled at the latest

$Prop_{ta}$ depends on the given problem, i. e. a task can have different properties that represent the information necessary to execute and fulfill the task. Exemplary properties are the kind of the task, the place of its occurrence, the amount of work to perform, or the costs of this task. t_{ta}^{start} respectively t_{ta}^{end} may remain unspecified, which means that ta may be executed from the start of the solution of the problem respectively fulfilled until its end. t_{ta}^{start} is not to be mixed up with the point in time ta is announced to the system respectively becomes available, denoted by $t_{ta}^{avail} \in Time$. This is the point in time when the agents in A get possibly informed about the presence of the task, which usually is before or at the latest at t_{ta}^{start}, more formally $\forall ta \in T : t_{ta}^{avail} \leq t_{ta}^{start}$.

If all information on the tasks of a problem is assumed to be deterministic and known a priori, i. e. $\forall ta \in T : t_{ta}^{avail} = 0$, the problem may be called *static*. If this information is gradually revealed over time, i. e. $\exists t_{ta_i}^{avail} : t_{ta_i}^{avail} < t_{ta_j}^{avail}$ for $i \neq j$ (usually there are more than just one such $t_{ta_i}^{avail}$), the problem may be called *dynamic*. If some of the data is random variables whose distributions are usually known, the problem may be called *stochastic*.

An instantiation of an abstract task ta for a specific problem P is denoted as ta^P. A particular problem instance then includes a set of tuples $(ta^P, t_{ta^P}^{avail})$, also called *events*, of instantiated tasks along with the corresponding times at which they become available to the agents. A *run instance* consists of several, interwoven problem instances. For example, a run instance may represent all the problem instances respectively tasks A has to solve respectively fulfill at a particular day. Consequently, a run instance is bounded by a maximal execution time in which the tasks have to be fulfilled.

Definition 6.2 (Run instance)
A run instance run is defined as a sequence

$$run^{P,Time} = ((ta_1^P, t_{ta_1^P}^{avail}), (ta_2^P, t_{ta_2^P}^{avail}), \ldots, (ta_n^P, t_{ta_n^P}^{avail}))$$

where

- P denotes the problem to be solved
- $Time$ is the maximal execution time in which run has to be solved

- $ta_i^P \in T^P$ is a instantiation of a task for P
- $t_{ta_i^P}^{avail} \in Time$ is the point in time ta_i^P becomes available to the agents, with $t_{ta_i^P}^{avail} \leq t_{ta_{i+1}^P}^{avail}$

The sequence of the tasks in a run instance is ordered according to the $t_{ta_i^P}^{avail}$. This may lead to different sequences if at least two tasks become available at the same time, i. e. if $\exists(ta_i^P, t_{ta_i^P}^{avail}), (ta_j^P, t_{ta_j^P}^{avail}) \in run : t_{ta_i^P}^{avail} = t_{ta_j^P}^{avail}, i \neq j$. However, this fact can be disregarded as defining an equivalence relation on the sequences that puts those sequences into the same equivalence class is simple. For the sake of readability, we will use a more simplified notation of a run instance.

Notation 6.1 (Run instance)
A run instance is notated as

$$run = ((ta_1, t_1), (ta_2, t_2), \ldots, (ta_n, t_n))$$

Usually, there will be a sequence of run instances that A has to solve. Such a sequence of run instances may represent all the problem instances A has to solve over several days, e. g. a week or a month. Based on the simplified notation of a run instance, a sequence of run instances of length k is then described as

$$\begin{aligned}(run_1, \ldots, run_k) \quad = \quad & (((ta_{11}, t_{11}), (ta_{21}, t_{21}), \ldots, (ta_{m_1 1}, t_{m_1 1})),\\ & ((ta_{12}, t_{12}), (ta_{22}, t_{22}), \ldots, (ta_{m_2 2}, t_{m_2 2})),\\ & \ldots,\\ & ((ta_{1k}, t_{1k}), (ta_{2k}, t_{2k}), \ldots, (ta_{m_k k}, t_{m_k k}))\end{aligned}$$

A *solution* for one particular run instance consists of several *assignments* of tasks to agents that handle these tasks.

Definition 6.3 (Assignment)
An assignment of a task ta_i to an agent Ag_j is defined as a triple (ta_i, Ag_j, t_k), where the task ta_i will be started by Ag_j at the point in time $t_k \in Time$.

The set of all possible assignments is denoted as $Assign$. Please note that fulfilling a task ta_i might require a sequence of actions by an Ag_j. Furthermore, depending on the application domain there may be additional restrictions, for instance that not every type of agent can perform every type of task. By the definition of assignments, we now can define a solution for a run instance.

Definition 6.4 (Solution)
A solution *sol* generated by a set of agents A for a particular run instance that consists of several tasks $\{ta_1, ta_2, \ldots, ta_m\}$ is defined as

$$sol(A, (ta_1, ta_2, \ldots, ta_m)) = ((ta'_1, Ag'_1, t'_1), \ldots, (ta'_m, Ag'_m, t'_m))$$

where

- $ta'_i \in \{ta_1, ta_2, \ldots, ta_m\}, \forall i \neq j : ta'_i \neq ta'_j$
- $Ag'_i \in A$
- $t'_i \leq t'_{i+1}, t'_i \in Time$

Please note that $\{ta_1, ta_2, \ldots, ta_m\}$ and $\{ta'_1, ta'_2, \ldots, ta'_m\}$ do not have to be related in any way, i. e. task do not have to be immediately started by one of the agents when they become available. This allows at least theoretically for the possibility that the agents in A can be more than purely reactive. The set of all possible solutions is denoted as Sol.

Usually, a solution is expected to be of a certain *quality*, regarding e. g. costs, time, used resources, etc., which is dependent on the particular problem of an application domain as well as on the agents in A solving the problem.

Definition 6.5 (Solution quality)
The quality $qual$ of a solution $sol \in Sol$ is defined as a function

$$qual : Sol \rightarrow \mathbb{R}^+$$

This is, if $qual(sol_1(A, (ta_1, \ldots, ta_m))) > qual(sol_2(A, (ta_1, \ldots, ta_m)))$, the solution sol_1 will solve a run instance consisting of tasks $\{ta_1, \ldots, ta_m\}$ better than the solution sol_2.

We assume that the agents that solve a problem respectively run instance behave according to an appropriate decentralized coordination mechanism (see Section 3.3). Usually, such a coordination mechanisms is realized in form of an algorithm, which is executed by each agent, or implemented by a set of simple local *rules*, which define the behavior of the agents, or as a mixture of both. The algorithms and rules may be specific to an agent, i. e. each agent may behave differently by e. g. following a different set of rules. Together, all the behaviors of the agents generate the emergent solution to a problem[1].

Therefore, the approach to change the overall behavior of the system in order to improve the efficiency of a solution, is to make changes to the local rule set of the agents, so that their local behavior can be adapted slightly. In particular, if an agent is only guided by an algorithm, as it is e. g. the case for IBC-based agents, it will be required to be capable of evaluating rules while it decides on the next steps to perform. These changes are made in terms of so-called exception rules, which extend the local rule set of an agent and apply only for specific situations.

[1] If the behavior of an agent is described by rules only, the application of the rules even *creates* the emergent behavior. But as this is part of the implementation details of the agent system, this aspect will not be regarded more thoroughly in this chapter.

Basically, exception rules may be realized as Event-condition-action (ECA) rules. ECA rules can be subsumed as reaction rules, which are in turn a special subclass of general rules. They automatically perform actions in response to events provided that the stated conditions hold. ECA rules allow an agent's reactive functionality to be defined and managed within a single rule base rather than being encoded in diverse algorithms, thus enhancing the modularity and maintainability of the agent's local rule base. An ECA rule has the general syntax

$$\text{on } \textit{event} \text{ if } \textit{condition} \text{ do } \textit{action(s)}$$

The event part specifies when the rule should be triggered, the condition part is a logical test that has to be satisfied, and the action part states the action(s) to be performed automatically if the condition holds. Executing a rule's action(s) may in turn trigger further ECA rules, and the rule execution proceeds until no more rules are triggered.

Definition 6.6 (Exception Rule)
An exception rule r for an agent Ag_i is defined as triple $r = (sit, dat, act)$, where $sit \in Sit_i$ represents the event, $dat \in Dat_i$ specifies a condition and $act \in Act_i$ is the action the agent has to perform in case that the condition holds.

The set of all possible rules is denoted as $Rule$. The evaluation of exception rules in turn implies an agent model in which the agents are capable of processing sensory information and choosing an action based on this information as well as the exception rules. This requires an extension of the agent definition (see Definition 2.2).

Definition 6.7 (Rule-applying agent)
A rule-applying agent Ag^r is defined as an extension of Ag such that

$$Ag^r = (Sit, Dat^r, Act, f^r_{Ag}, cr^r_{Ag})$$

where

- Sit is the set of situations Ag^r can be in
- Dat^r is the extension of Dat with rules
- f^r_{Ag} is the extension of f_{Ag} for applying rules
- $cr^r_{Ag} : 2^{Rule} \rightarrow Act$ is a conflict resolution mechanism that decides which rule(s) to apply in case more than one rule is applicable

Exception rules are influencing the agent's decision making process by changing the action f^r_{Ag} generates. The decision function of a rule-applying agent hence has the form

$$f^r_{Ag}(sit, dat) = \begin{cases} f_{Ag}(sit, dat) & \text{if } \neg\exists r \in Dat^r : eval(sit, dat) = true \\ cr^r_{Ag}(r_1, r_2, \ldots r_n) & \forall r_i \in Dat^r : eval(sit, dat) = true \text{ otherwise} \end{cases} \tag{6.1}$$

where *eval* evaluates the occurrence of an event along with a respective condition to *true* or *false*.

6.2 Generic Advisor Model

Admittedly, there exist some instances of the general problem and run structure described in the previous section for which it is easy or at least possible to generate optimal solutions, for example in the case of static problems. But for most instances, in particular in the case of dynamic problems, which we are interested in, it is very difficult or even impossible. For example, finding an optimal solution might be an NP-complete problem for a particular application. If a task $ta_i \in T$ can arrive at any point in time within $Time$, the requirement that all tasks need to be started within $Time$, which is usually accompanied with the additional requirement that all tasks need also to be fulfilled within $Time$, will often lead to non-optimal solutions. Reasons for this non-optimality are listed in Section 5.1. As a consequence, the design of self-organizing emergent systems focuses rather on beneficial properties such as flexibility, scalability, and robustness than on optimality and predictability (see also Chapter 3 and 4). Nevertheless, the quality of the solution, in particular in industrial settings, remains of high importance, too.

In this section, we therefore present an approach that enhances a self-organizing emergent MAS for the stated type of problems by the concept of an EIA. The EIA allows for a better solution quality over a sequence of run instances, while preserving the beneficial properties of the basic system. Due to the characteristics of self-organizing emergent systems as well as the resulting constraints and requirements for their adaptation (see Section 5.4), there are three premises that must hold for the application of the EIA concept:

1. Each agent in A must be able to collect data about its local behavior, i. e. its sensory input and its actions, and must be able to "dump" this history to a central collection unit at least once during a run instance.

2. Each agent in A participating in the self-organizing emergent solution to a problem can be extended to a rule-applying agent, i. e. the decision function f_{Ag} of each agent can be extended to deal with rules, which will be stored in the agents' internal data areas.

3. A sequence of run instances must have a (sub)set of similar tasks in (nearly) each run instance of the sequence.

The first premise is required to account for the low observability and poor controllability of a self-organizing emergent system. Because due to these constraints a pull strategy applied by the EIA in order to collect enough information is in general not possible, this premise allows for a push strategy to be applied. Usually, this information transfer may happen at one designated location or point in time, e. g. at the end of a run instance or when a moving agent returns to a central location. At the same time, the agents' rule sets may be updated.

The second premise is required to allow an adaptation of the local behavior of an agent by the EIA. An agent is not required to act based on rules only but arbitrary algorithms may be used for f_{Ag} as well, as long as the integration and evaluation of rules is supported. If the local behavior of an agent is already based on rules, this premise will not be required.

While the first two premises can be achieved very easily, the third premise seems to be very restrictive. However, in everyday life there are many problems that fulfill this premise. For instance, delivery companies usually have daily recurring tasks together with one-of-a-kind tasks. This premise enables the EIA to focus its attention to recurring problems that can be observed repeatedly and allows the EIA to 'look into the future' as well as to optimize the agents to work under these relevant conditions. Of course, recurring tasks are allowed to change over time.

Based on these premises, the EIA may be realized as an agent as well, however with extended capabilities compared to the agents of the basic controlled system. Consequently, an *advised MAS* can be defined as an extension of a MAS (see Definition 2.3), which is under the control of a special advisor agent Ag_{EIA}.

Definition 6.8 (Advised multi-agent system)
An advised multi-agent system $advMAS$ is defined as

$$advMAS = (A \cup \{Ag_{EIA}\}, Env)$$

where

- A is a set of basic agents $Ag_1, \ldots, Ag_m$
- Ag_{EIA} is an advisor agent
- Env is a common environment (or at least parts of it) the agents in A share in order to interact with each other

In general, the advisor is not an element of the agent set A of the basic MAS itself, but rather an additional element (see Figure 6.1) and hence does not behave according to the applied coordination mechanism of the basic MAS. This allows the basic MAS to run with or without the advisor, maintaining the reliability of the overall solution.

The EIA is able to act autonomously, i. e. it improves the efficiency of the basic MAS without any user interaction. Basically, the advisor therefore implements the four activities of a closed feedback control loop (see Figure 5.4), however, separated

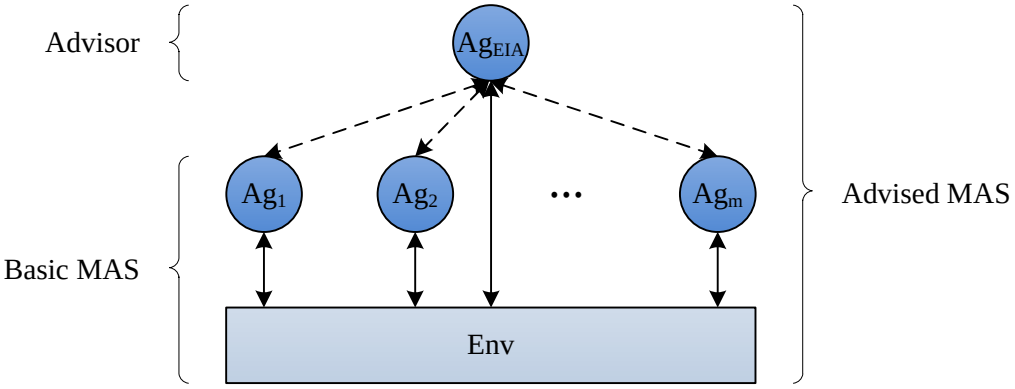


Figure 6.1: An advised multi-agent system

into six distinct functions, which are connected by one data model (see Figure 6.2). All of these functions will be described in more detail in the subsequent subsections:

1. ***Receive* local agent histories:** The advisor collects the local histories of the advised agents, i. e. mainly the situations they have perceived and the actions they have performed as well as information about the environment, at least once during a run instance or at its end. This is essential, as due to the low observability of the system the advisor usually is neither aware of the tasks that have to be fulfilled during a run instance nor the actions executed by the agents of the system. The advisor stores the collected local history of each agent in its data model. The interaction between the advisor and the agents only occurs when communication is possible and does not interfere with the fulfillment of the agents' tasks (see Subsection 6.2.1).

2. ***Transform* local agent histories into global history:** Based on the received local agent histories the advisor then creates the global history of the system (and the system environment as far as possible). This provides the advisor with a global view on the past sequence of run instances the agents had to solve so far (see Subsection 6.2.2).

3. ***Extract* recurring tasks from global history:** Based on the global system history, more specifically the sequence of run instances, the advisor then identifies recurring tasks in these run sequences. Because in general tasks may slightly change over time, i. e. between two or more run instances, the advisor not only identifies tasks that are identical in all run instances but also tasks for which similar tasks exist in all or at least most of the run instances. This set of recurring tasks apparently constitutes a core problem that appears repeatedly in the system (see Subsection 6.2.3).

4. ***Optimize* solution of recurring tasks:** The advisor then calculates a (nearly) optimal solution for the set of recurring tasks by means of standard

optimization algorithms. An optimization of the solution of all tasks that occur in the run instances is not necessary, because usually not all of these tasks will occur in future run instances again. By contrast, it can be supposed that at least the recurring tasks (or similar ones) will still occur in future (see Subsection 6.2.4).

5. ***Derive* rules from the optimal solution:** If the emergent solution for the set of recurring tasks is much worse than the calculated optimal solution for the set of recurring tasks, the advisor will derive rules (advices) for the agents that do not behave optimally with regard to this optimal solution. Therefore, the advisor first identifies and extracts the solution the emergent system has generated for this set of recurring tasks, determines differences between the emergent solution and the optimal solution, and, if applicable, creates from these differences rules for the misbehaving agents (see Subsection 6.2.5).

6. ***Send* derived rules to the agents:** Finally, the advisor transfers newly created rules to the agents the next time it can exchange information with them. From the moment the agent has stored the new rule, the rule will be incorporated into the agents' decision mechanism and improve the overall quality of the solution (see Subsection 6.2.6).

Apparently, concrete realizations of these functions depend on the application at hand and on the realization of the basic MAS including their coordination principles. In order to support these functions, the data model of the EIA fulfills the following requirements:

- It covers all basic events that can occur in the basic system. Furthermore, it describes the environment in a way that contains all knowledge needed by any of the above functions. Thus, the data model covers tasks as well as more simple, low-level building blocks.

- It provides input to all functions, because all functions store their intermediate results in the data model. Thereby, the representation of data is an important aspect, in particular for the functions *extract* and *optimize*. Thus, the tasks and the assignments of the agents to these task are represented carefully.

Figure 6.3 shows the general interaction schema between an agent Ag_i of the basic MAS and the advisor agent Ag_{EIA}. As presumed, the advisor agent has to be able to communicate with each Ag_i for both receiving local histories and sending derived rules. An agent Ag_i is able to proceed its work during the calculations of Ag_{EIA}, i. e. it is not blocked until the update of its rule set. The figure might be a little bit deceiving with regard to the time Ag_{EIA} requires to perform each function as well as how much time can be between the derivation of rules and their sending to Ag_i. This depends significantly on the complexity of the problem to be solved, the instantiation of the functions, and the capabilities of the hardware infrastructure.

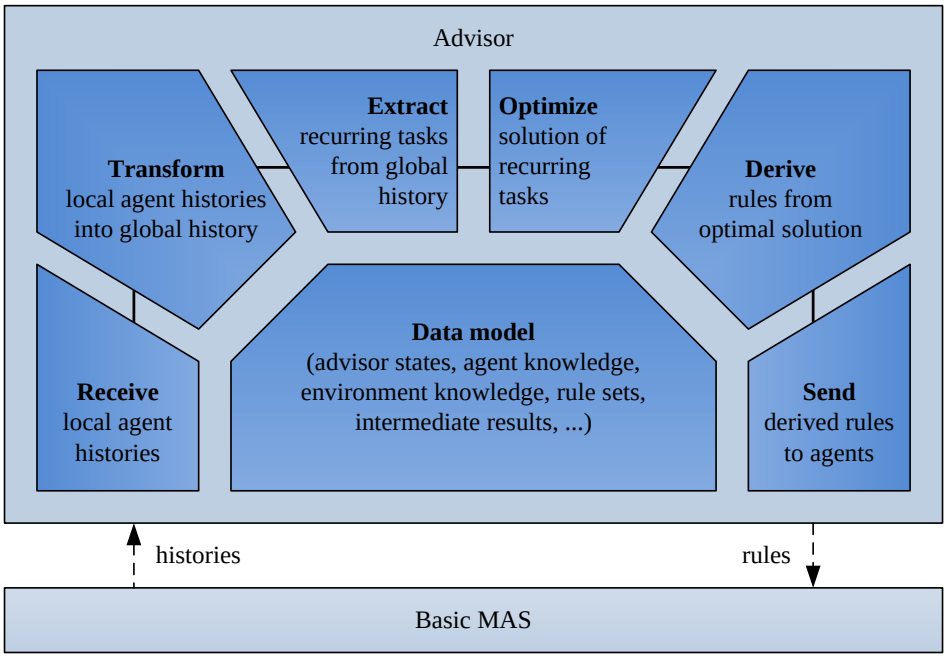


Figure 6.2: Functional architecture of an advisor

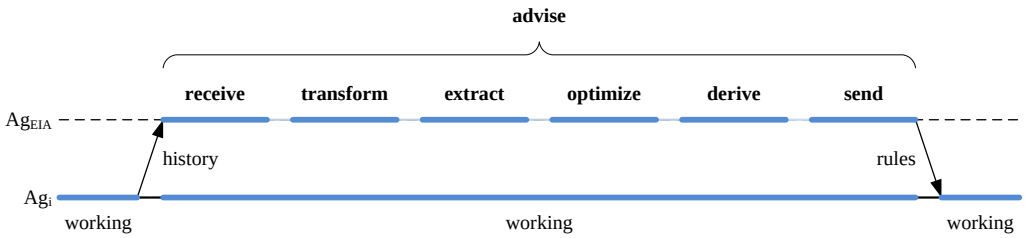


Figure 6.3: Interaction schema between an agent Ag_i and the advisor Ag_{EIA}

More formally, the advisor agent Ag_{EIA} can be defined as an instantiation of an abstract agent (see Definition 2.2):

Definition 6.9 (Efficiency Improvement Advisor)
The efficiency improvement advisor (EIA) is defined as a special agent

$$Ag_{EIA} = (Sit_{EIA}, Act_{EIA}, Dat_{EIA}, f_{Ag_{EIA}})$$

where

- Sit_{EIA} is a set of situations describing possible messages and included data received from the agents in A
- Act_{EIA} is the set of actions Ag_{EIA} can perform[2], more specifically *receive*, $transform$, *extract*, *optimize*, *derive*, and *send*
- Dat_{EIA} contains the set of possible states of the Ag_{EIA}'s knowledge about the advised agents and the environment they operate in, a set of rules Ag_{EIA} can create, and structures for all intermediate results that are created by the actions of Ag_{EIA}
- $f_{Ag_{EIA}} : Sit_{EIA} \times Dat_{EIA} \rightarrow Act_{EIA}$ is the decision function Ag_{EIA} uses to determine its next action

The actions *receive*, $transform$, *extract*, *optimize*, *derive*, and *send* are all part of a super-action *advise*. This super-action performs these actions in a sequential order. Because each of the actions is able to store the results of its computation in the data model Dat_{EIA}, the intermediate results are available to subsequent actions. *Advise* may be performed either when the Ag_{EIA}'s situational description contains a new local history from one of the agents of the basic MAS or at distinct points in time, e. g. at the end of a run instance when histories of several agents are received.

Subsequently, in the next six subsections we will describe each of the actions in more detail, whereas Subsection 6.2.7 provides more details about the data model of Ag_{EIA}.

6.2.1 Receive Local Agent Histories

The main task of the action $receive \in Act_{EIA}$ is to collect the local histories of the agents and to store them in the internal data structure Dat_{EIA} of the Ag_{EIA}, so that the data included in the histories can be used by subsequent actions. The latter require mainly knowledge about two different types of data:

- **Environment data:** Data about the environment, i. e. a map of the environment structure, can be either available to Ag_{EIA} already at its start or extracted from the histories of the agents in A, which depends mainly on the

[2]Most actions of Ag_{EIA} are composed of smaller sub-actions, which are again part of Act_{EIA}.

constraints of the given problem. If the environment structure is already well known a priori, the agents in A as well as Ag_{EIA} might be equipped with a sufficient representation of the environment even before they start to work. An exemplary representation might be a directed graph, which contains edges between discrete points. Apparently, a detailed representation of the environment is more advantageous because it enables Ag_{EIA} to leverage specific features of the environment better. If a representation of the environment is not available a priori, e. g. in dynamic problems in which even the environment structure may change over time, the agents will have to create an internal representation themselves.

The local history of an agent $Ag_i \in A$ thus has to include information about all situations $sit \in Sit_i$ it has perceived during its work, because according to the definition of an element sit of Sit (see Subsection 2.4.2) information about the environment states s_{env} can be extracted from it. Data about the environment will be stored in $dat_{EIA}^{env} \in Dat_{EIA}^{Env}$.

- **Agent data:** Data about the local behaviors of the basic agents, i. e. their perceptions and the actions they have performed at each point in time, has to be initially collected by the agents themselves during the execution of the system. Thus, beside information on Sit_i the history of an agent Ag_i also has to include information on all actions $act \in Act_i$ the agent has performed, ordered by the point in time they have been performed, as well as information about the corresponding states of Dat_i. This information will allow to deduce a global view on the system history.

Based on these two types of required data, the local agent history can be defined.

Definition 6.10 (Local agent history)
The history $hist$ of an agent $Ag_i \in A$ is defined as

$$hist_i = ((sit_i^t, dat_i^t, act_i^t), (sit_i^{t+1}, dat_i^{t+1}, act_i^{t+1}), \ldots, (sit_i^{t+k}, dat_i^{t+k}, act_i^{t+k}))$$

where

- $sit_i^l \in Sit_i$ is the agent's situation at the point in time l
- $dat_i^l \in Dat_i$ is the data stored in the agent at the point in time l
- $act_i^l \in Act_i$ is the action the agent performed at the point in time l
- $t-1$ is the point in time at which the history of Ag_i was made available to Ag_{EIA} for the last time respectively the point in time before the system has started

The set of all possible histories that can be received from an agent Ag_i is denoted as $Hist_i$. All histories received from Ag_i are stored in $dat_{EIA}^{hist_i}$, that is

$$dat_{EIA}^{hist_i} = ((sit_i^1, dat_i^1, act_i^1), (sit_i^2, dat_i^2, act_i^2), \ldots, (sit_i^o, dat_i^o, act_i^o)) \tag{6.2}$$

where o denotes the last point in time included in a history of Ag_i, which possibly may be the current point in time. The set of all possible values for $dat_{EIA}^{hist_i}$ is denoted as $Dat_{EIA}^{Hist_i}$. A "dump" of the history $hist_i \in Hist_i$, i. e. a message from an agent Ag_i to Ag_{EIA} including $hist_i$, is part of the situations Sit_{EIA} the advisor Ag_{EIA} can perceive. The history becomes part of $dat_{EIA}^{hist_i}$ by invoking the action $receive \in Act_{EIA}$.

Definition 6.11 (Receive)
The reception of a local history of an agent Ag_i and its storage in the internal data structure is a function defined as

$$receive : Hist_i \times Dat_{EIA}^{Hist_i} \rightarrow Dat_{EIA}^{Hist_i}$$

where

- $Hist_i$ is the set of all possible histories that can be received from an agent Ag_i
- $Dat_{EIA}^{Hist_i}$ is the set of all possible histories received from an agent Ag_i stored in Dat_{EIA}

The action *receive* chooses a value from $dat_{EIA}^{hist_i} \in Dat_{EIA}^{Hist_i}$ that represents the histories received from Ag_i so far combined with the history $hist_i$ just received and stores this value in the internal data structure again. The set of all possible histories that can be received from all agents in A with $m = |A|$ is denoted as $Hist_A$, that is

$$Hist_A \subseteq Hist_1 \times Hist_2 \times \ldots \times Hist_m \tag{6.3}$$

All histories received from the agents in A are stored in $dat_{EIA}^{hist_A}$, that is

$$\begin{aligned} dat_{EIA}^{hist_A} &= (dat_{EIA}^{hist_1}, dat_{EIA}^{hist_2}, \ldots, dat_{EIA}^{hist_m}) \\ &= (((sit_1^1, dat_1^1, act_1^1), (sit_1^2, dat_1^2, act_1^2), \ldots, (sit_1^{o_1}, dat_1^{o_1}, act_1^{o_1})), \\ &\quad ((sit_2^1, dat_2^1, act_2^1), (sit_2^2, dat_2^2, act_2^2), \ldots, (sit_2^{o_2}, dat_2^{o_2}, act_2^{o_2})), \\ &\quad \ldots, \\ &\quad ((sit_m^1, dat_m^1, act_m^1), (sit_m^2, dat_m^2, act_m^2), \ldots, (sit_m^{o_m}, dat_m^{o_m}, act_m^{o_m}))) \end{aligned} \tag{6.4}$$

where $m = |A|$ and $o_1, o_2, \ldots, o_m$ denote the last points in time included in the corresponding histories. The set of all possible values for $dat_{EIA}^{hist_A}$ is denoted as $Dat_{EIA}^{Hist_A}$.

6.2.2 Transform Local Agent Histories into Global History

The main task of the action $transform \in Act_{EIA}$ is to deduce the global history of the system out of the received local histories of the agents, in order to provide Ag_{EIA} with a global view on the past run instances. Thereby, not a full reproduction of the ordered system events is done, but rather certain details of the occurred problems and their solutions are of importance. More specifically, the subsequent actions of Ag_{EIA} require knowledge about the tasks that had to be solved during the run instances and the assignments of these tasks to the agents of the basic system.

Thus, the starting point for Ag_{EIA} is the deduction of the global history of single agents first. The global history of a single agent is deduced from a combination of its perceptions and actions included in the local history of the agent[3]. The global history consists of a sequence of tasks that has been handled by the agent during the run instances so far.

Definition 6.12 (Global agent history)
The global history $ghist$ of an agent Ag_i is defined as a sequence

$$ghist_i = (ta_1, ta_2, \ldots, ta_k)$$

where $ta_i \in T$ and k is the number of tasks Ag_i has handled.

The global history of agent Ag_i is stored in $dat_{EIA}^{ghist_i} \in Dat_{EIA}^{GHist_i}$, where $GHist_i$ denotes the set of all possible tasks in each possible run instance for an agent Ag_i. An assignment of an agent Ag_i to a task ta_j at time t_l exists, if and only if ta_j is an element of the global agent history, i. e. the agent Ag_i performed ta_j in any run instance. Distinguishing in which run instance a task appeared is not necessary at this point. The global agent histories will be used to extract recurring tasks and this process uses all the tasks identified so far as input, regardless of the run instance they appeared in.

Definition 6.13 (Deduce)
The deduction of the global agent history $ghist_i$ out of the received local agent histories of an agent Ag_i is a function defined as

$$deduce : Dat_{EIA}^{Hist_i} \rightarrow Dat_{EIA}^{GHist_i}$$

where

- $Dat_{EIA}^{Hist_i}$ is the set of all possible histories received from an agent Ag_i stored in Dat_{EIA}
- $Dat_{EIA}^{GHist_i}$ is the set of all possible tasks in each possible run instance for an agent Ag_i stored in Dat_{EIA}

[3]Depending on the representation of data in the agent system, further transformations may have to be applied first.

The deduction of the global agent history is a necessary sub-action of $transform$. However, because a single agent usually did not perceive or handle all tasks on its own, e. g. due to its positions or capabilities, the global histories of all agents are required to generate the global history of the system.

Definition 6.14 (Global history)
The global history $GHist$ of a set of agents $A = Ag_1, \ldots, Ag_m$ is defined as

$$GHist_A \subseteq GHist_1 \times GHist_2 \times \ldots \times GHist_m$$

where $GHist_i$ is the set of all possible tasks in each possible run instance for an agent Ag_i.

Whenever Ag_{EIA} has the possibility to optimize the efficiency of the basic system, the action $transform \in Act_{EIA}$ transforms these global agent histories into a global system history and stores the latter in its internal data model.

Definition 6.15 (Transform)
The transformation of the global agent histories $Dat_{EIA}^{GHist_1}, \ldots, Dat_{EIA}^{GHist_m}$ with $m = |A|$ to the global history of the system $Dat_{EIA}^{GHist_A}$ stored in Dat_{EIA} is a function defined as

$$transform : Dat_{EIA}^{GHist_1} \times Dat_{EIA}^{GHist_2} \times \ldots \times Dat_{EIA}^{GHist_m} \rightarrow Dat_{EIA}^{GHist_A}$$

The process of transformation may be complicated, dependent on the collected histories and the needs of the later actions of the advisor. The result of this process essentially contains the sequence of run instances $(run_1, \ldots, run_k)$, see page 151, the system has solved so far, i. e. the tasks the agents have handled.

6.2.3 Extract Recurring Tasks from Global History

The main task of the action $extract \in Act_{EIA}$ is the identification of recurring tasks in the sequence of run instances the previous action has reconstructed. In principle, there are several ways to find recurring tasks in a sequence of run instances. Unfortunately, for many applications the problem is more complicated than just finding tasks that appear in each run instance. Ag_{EIA} not only has to identify tasks that are identical in all run instances but also tasks for which similar tasks exist in all or at least most of the run instances. Thus, data from several run instances has to be used for the extraction. In general, differences between tasks can occur in each of the attributes of a task. For instance, the task of delivering a parcel to a particular house in a street at midday is usually not very different from delivering it to a neighboring house in the early afternoon, so that having one delivery each day to one of the two houses should put this task into the sequence of recurring tasks. Apparently, the possibility of finding a recurring, totally identical tasks in such scenarios is very small.

The extraction of recurring tasks in a sequence of run instances thus requires two steps: first, to find *task patterns*, i. e. tasks that appear in the run instances and that are very similar. Second, to identify and distill those tasks that appear repeatedly, indicating that they are of higher importance, which makes the solution of these tasks worthwhile to optimize. An intuitive approach to identify tasks patterns would be to apply a pattern recognition algorithm on the tasks of each run instance and to accumulate afterwards the identified patterns of several run instances to find recurring tasks within all those patterns. Unfortunately, pattern recognition and accumulation are very time intensive. Moreover, to specify an appropriate algorithm, to compare the resulting task patterns, and to identify relevant recurring tasks, the representation of tasks plays a major role. However, it is neither obvious, what the representation has to be like for a sequence of tasks, how the representation can be stored for subsequent actions of the advisor, nor how it can be used to calculate other relevant data. To use a sequence of tasks in subsequent actions, an identified sequence has to be taken apart and split into single tasks again.

Thus, a more reasonable approach to identify task patterns is *clustering* [JMF99]. The goal of clustering is to separate a finite, unlabeled set of data into a finite, discrete set of "natural", hidden data structures [XW05]. Thus, tasks that are very similar should be combined to the same task pattern respectively cluster of tasks.

Definition 6.16 (Cluster of tasks)
A cluster of tasks $Cl = (T^P, ce)$ consists of a set of tasks $T^P \subseteq T$ and a centroid ce over these tasks.

A centroid is the mathematical median of the elements of a cluster, but has not to be an element of the cluster itself. The centroid is required in order to determine, if a task is similar enough to become part of a given cluster. The determination of the centroid itself depends on the description of the tasks and is accomplished by a centroid function.

Definition 6.17 (Centroid function)
Let T be the set of tasks. The centroid of a cluster of tasks is determined by a function $cent : 2^T \rightarrow T$.

In order to determine, if a task is similar enough to another task, which may be a centroid as well, a measure of proximity becomes important. Almost all clustering algorithms are explicitly or implicitly connected to some definition of proximity, which is a generalization of similarity and dissimilarity. We base the proximity of tasks on their similarity.

Definition 6.18 (Similarity function)
Let T be the set of tasks. The similarity of two tasks is determined by a function $sim : T \times T \rightarrow \mathbb{R}$.

Apparently, the similarity again depends on the description of the tasks and the problem to solve. It has to ensure that tasks that are very similar with regard to

their attributes have a high similarity value, while tasks that differ greatly have a lower similarity value. Based on the two functions *cent* and *sim* defined above, the clustering of a set of tasks can now be defined in general.

Definition 6.19 (Clustering of tasks)
Given a set of tasks $T' = \{ta_1, \ldots, ta_{m_k}\}$ with $ta_i \in T$, the clustering of T' is defined as the partitioning of T' in x clusters $Cl_1, \ldots, Cl_x$ with $x \leq |T'|$ such that

1. $Cl_i \neq \emptyset, i \in \{1, \ldots, x\}$
2. $\bigcup_{i=1,\ldots,x} Cl_i = T'$
3. $Cl_i \cap Cl_j = \emptyset, i, j \in \{1, \ldots, x\}$ and $i \neq j$

The concrete clustering of tasks by Ag_{EIA} starts as soon as a sufficient amount of data has been received and transformed, i. e. after a predefined number of runs have passed that serve as *learning* time. For the clustering, a clustering algorithm is required that clusters the tasks included in the global history of the system $Dat_{EIA}^{GHist_A}$. The clustering is hence a necessary sub-action of *extract*.

Definition 6.20 (Cluster)
Let T be the set of tasks and $Dat_{EIA}^{GHist_A}$ the global history of the set of agents A stored in Dat_{EIA}. The clustering of tasks is a function defined as

$$cluster : Dat_{EIA}^{GHist_A} \rightarrow 2^{2^T}$$

Assuming that $Cl_1, \ldots, Cl_x$ is the result of the clustering function respectively algorithm, the next step is to determine all clusters that are big enough to indicate that they contain recurring tasks. Having k run instances, these are all clusters Cl_i with $|Cl_i| \geq minocc \cdot k$. $minocc$ is a given user determined threshold parameter, with $0 < minocc \leq 1$, which defines the minimum number of runs a task has to occur in to be recurring. If $Cl'_1, \ldots, Cl'_y$ are all clusters fulfilling this condition, then all $cl'_j \in Cl'_i$ with $sim(cl'_j, ce'_i)$ is maximal will be distilled and put into the set of recurring tasks T^{rec}. These cl'_j are called medoids. A medoid is similar in concept to a centroid, but in contrast to a centroid, a medoid is always an element of a cluster.

Definition 6.21 (Medoid)
The medoid cl_j of a cluster of tasks Cl_i with centroid ce_i is a task such that $\forall cl_l \in C : sim(cl_j, ce_i) \geq sim(cl_l, ce_i)$.

If there are Cl'_i with $|Cl'_i| \geq (1+minocc) \cdot k$, the cluster will represent a task pattern that occurs more than once during a run instance, maybe even several times. In this case also the $cl''_j \in Cl'_i \setminus \{cl'_j\}$ with $sim(cl''_j, ce'_i)$ is maximal will be distilled and put into T^{rec}. This process continues for $|Cl'_i| \geq (2 + minocc) \cdot k$, etc. This distillation of medoids into the set of recurring tasks, which are stored in the data structure $dat_{EIA}^{rec} \in Dat_{EIA}^{Tasks}$, is the second necessary sub-action of *extract*.

Definition 6.22 (Distill)
Let T be the set of tasks and Dat_{EIA}^{Tasks} the set of of all possible sets of recurring tasks stored in Dat_{EIA}. The distillation of tasks is a function defined as

$$distill : 2^{2^T} \rightarrow Dat_{EIA}^{Tasks}$$

The set of distilled recurring tasks T^{rec} is then sorted according to the time the tasks started. The result is hence a sorted sequence of recurring tasks.

In summary, the action $extract \in Act_{EIA}$ is composed of two sub-actions: First, applying a clustering algorithm to the reconstructed tasks in $Dat_{EIA}^{GHist_A}$ to identify recurring task patterns, and second, distilling the medoids as the most relevant recurring tasks from these clusters and storing them in the data structure dat_{EIA}^{rec}.

Definition 6.23 (Extract)
The extraction of recurring tasks is a function defined as

$$extract : Dat_{EIA}^{GHist_A} \rightarrow Dat_{EIA}^{Tasks}$$

where

- $Dat_{EIA}^{GHist_A}$ is the set of all possible global histories of a set of agents A stored in Dat_{EIA}
- Dat_{EIA}^{Tasks} is the set of all possible sets of recurring tasks stored in Dat_{EIA}

Thus, instead of identifying relevant tasks and clustering them, the tasks are clustered first and relevant tasks are then distilled from these clusters. These recurring tasks constitute a core problem that appear repeatedly in the system and serves as input for the optimization process. It should be noted that for applications where the recurring tasks may change over time, this approach for realizing *extract* should not use all k run instances from the beginning of A's work, since most probably after some time the set of recurring tasks will become very small or even empty. In such cases, a parameter k_{max} should be defined and only the run instances $run_{k-k_{max}}, run_{k-k_{max}+1}, \ldots, run_k$ should be used for the clustering. Moreover, k_{max} should also be used in the conditions using *minocc* for identification. While a change in the recurring tasks obviously will not immediately be noticed (which would require to really being able to look into the future), at the latest k_{max} run instances after the change Ag_{EIA} will be aware of the change and will create new advice for the agents in A. Naturally, if the recurring tasks change faster than k_{max} run instances, Ag_{EIA} will not be able to detect recurring tasks very well and A will have to rely on the basic decision making of its agents without advice.

6.2.4 Optimize Solution of Recurring Tasks

The main task of the action $optimize \in Act_{EIA}$ is to calculate a (nearly) optimal solution for the set of recurring tasks extracted by the previous action. Optimizing

the solution of all tasks that occur in the run instances is needless, because we cannot assume that all these tasks will occur in future again. However, we can assume that at least the recurring tasks identified by the previous action will still occur in future.

Please note that at this time all data of the problem, i. e. the set of recurring tasks, is known. Thus, the problem shifted from a dynamic one (as it was at runtime) to a static one. This requires Ag_{EIA} to have an integrated optimization algorithm that optimizes the solution to the static optimization problem representing the dynamic problem that A tries to solve.

Definition 6.24 (Solution optimization)
Let $T' \in 2^T$ be a set of tasks forming a problem P, *sol* a solution to P, and *qual* a solution quality function. The optimization of *sol* is a function defined as

$$opt : 2^T \rightarrow 2^{Sol}$$

such that the following conditions hold for every $Sol' \in 2^{Sol}$:

1. $\forall sol' \in Sol', \forall sol'' \in 2^{Sol} \setminus Sol' : qual(sol') > qual(sol'')$
2. $\forall sol', sol'' \in Sol' : qual(sol') = qual(sol'')$

Although such an optimization algorithm does not have to look into the future, the static optimization problem still can be very difficult to solve. An optimal solution sol_{opt} is then one of the elements of the set of optimal solutions calculated by *opt*. The optimal solution sol_{opt}^{rec} for a set of recurring tasks $(ta_1, ta_2, \ldots, ta_k)$ handled by a set of agents A is stored in $dat_{EIA}^{sol_{opt}}$. The set of all possible optimal solutions for a set of recurring tasks and a set of agents is denoted as $Dat_{EIA}^{Sol_{opt}}$. This allows for the definition of the action *optimize*.

Definition 6.25 (Optimize)
The optimization of a solution to a problem described by a set of recurring tasks is a function defined as

$$optimize : Dat_{EIA}^{Tasks} \rightarrow Dat_{EIA}^{Sol_{opt}}$$

where

- Dat_{EIA}^{Tasks} is the set of all possible sets of recurring tasks stored in Dat_{EIA}
- $Dat_{EIA}^{Sol_{opt}}$ is the set of all possible optimal solutions for a set of recurring tasks and a set of agents stored in Dat_{EIA}

The proximity of the calculated solution to a really optimal solution depends mainly on the quality of the optimization algorithm, the computational complexity of the problem, and the time for computation. If the time for computation is too short, only searching for a nearly optimal solution for the set of recurring tasks instead of the optimal one may be more appropriate. However, we will only speak of an optimal solution, even if this notion may not be correct for all instances.

6.2.5 Derive Rules from Optimal Solution

The main task of the action $derive \in Act_{EIA}$ is the derivation of rules from the optimal solution for the set of recurring tasks calculated by the previous action, which can then be sent to the agents for improving their solution, if necessary. Therefore, Ag_{EIA} has to perform four essential steps, constituting sub-actions of $derive$:

1. $Identify$ the solution of the emergent system for the set of recurring tasks
2. $Assess$ the quality of the emergent solution
3. $Determine$ differences between the emergent solution and the optimal solution, if the emergent solution is much worse than the optimal solution
4. $Create$ rules for the agents that do not behave optimally

6.2.5.1 Identification of Emergent Solution

The sub-action $identify \in Act_{EIA}$ is used to identify from the global history of the system $GHist_A$ generated by the action $transform$ the solution $sol_{emerg} \in Sol^P$ the emergent system has produced for the set of recurring tasks stored in dat^{rec}_{EIA} at last. The action yields unordered assignments for all of the tasks in the set of recurring tasks of the form $((ta_1, Ag_1, t_1), (ta_2, Ag_2, t_2), \ldots, (ta_k, Ag_k, t_k)$, which can then be put together to create the emergent solution.

Definition 6.26 (Identify)
The identification of the emergent solution sol_{emerg} for a set of recurring tasks is a function defined as

$$identify : Dat_{EIA}^{GHist_A} \times Dat_{EIA}^{Tasks} \rightarrow Dat_{EIA}^{Sol_{emerg}}$$

where

- $Dat_{EIA}^{GHist_A}$ is the set of all possible global histories of a set of agents A stored in Dat_{EIA}
- Dat_{EIA}^{Tasks} is the set of all possible sets of recurring tasks stored in Dat_{EIA}
- $Dat_{EIA}^{Sol_{emerg}}$ is the set of all possible sets of emergent solutions of a set of agents stored in Dat_{EIA}

However, identifying the emergent solution created by A for the recurring tasks is not trivial. Very often, there will be other tasks mixed into fulfilling the recurring tasks, or in the last run instance not all of the recurring tasks might have occurred, as the size of the clusters representing the recurring tasks can be smaller than k. The fact that the agents fulfill other tasks while fulfilling the recurring tasks means that Ag_{EIA} cannot determine the quality based on measuring what really happened. For

example, between fulfilling two tasks of the recurring task set in a transportation domain, a vehicle agent might have to drive to a far off location to fulfill a not recurring task in a particular run instance. Adding the traveled distance of this agent between the two recurring tasks to the travel cost (if this is the quality criterion) would worsen the emergent solution, although in other run instances the recurring tasks are solved well.

But for such an application a lower bound for the costs that would emerge can be provided, if there were no additional tasks, which is the distance the agent has to travel after the first recurring task to start performing the second one. Such lower bounds are possible to be determined for many applications and many quality criteria. If the quality of such a lower bound is far from the optimum, then advice from Ag_{EIA} will be useful for many run instances.

Given that the last sol_{emerg} and its quality are already an approximation, the problem of a recurring task not occurring in the last run instance can now be solved, too. Ag_{EIA} determines, which agent fulfills the task in the sol_{emerg} by going back one more run instance (or several). This also allows to determine the correct position of the task in the sequence of tasks the agent performs.

6.2.5.2 Assessing the Solution Quality

In order to determine, if an optimization of the emergent solution is necessary and worthwhile, the sub-action $assess \in Act_{EIA}$ compares the qualities of both the emergent solution and the optimal solution. If the emergent solution is of an acceptable quality compared to the optimal one, i. e. $qual(sol_{emerg})/qual(sol_{opt}) > qualthresh$ for a user defined quality threshold $0 < qualthresh \leq 1$, an optimization of the emergent solution is not worthwhile, e. g. if the emergent solution is within 90% of the optimal one, and the work of Ag_{EIA} is done until new information arrives. Otherwise, Ag_{EIA} has to optimize the emergent solution. $qualthresh$ depends on the concrete application domain and should be carefully evaluated.

6.2.5.3 Determination of Differences Between Solutions

In case that an optimization of the emergent solution is necessary, the sub-action $determine \in Act_{EIA}$ compares the differences between both the emergent and the optimal solution. Therefore, the identified assignments in both solutions are sorted by the order in which the tasks are performed in the solution. This sorting yields the solutions sol'_{opt} and sol'_{emerg}. The sorted solutions are then compared sequentially to find the first assignment that differs in both solutions. Given that $sol'_{opt} = ((ta^1_1, Ag^1_1, t^1_1), \ldots, (ta^1_p, Ag^1_p, t^1_p))$ and $sol'_{emer} = ((ta^2_1, Ag^2_1, t^2_1), \ldots, (ta^2_p, Ag^2_p, t^2_p))$ then Ag_{EIA} identifies the assignment $j : ta^1_j \neq ta^2_j \vee Ag^1_j \neq Ag^2_j$ such that $\forall i < j : ta^1_i = ta^2_i \wedge Ag^1_i = Ag^2_i$. Since both solutions are sorted according to the t_l-values, this is essentially the first assignment of a task to an agent for which the agents in A deviated from the optimal solution for the recurring tasks.

Please note that the t_l-values are currently not incorporated in the comparison due

to several reasons: First, the order of the optimal solution is usually not completely different from the order of the actual solutions. In many cases, the order is really only different because the tasks were served by different agents (a suboptimal behavior) and therefore will be correct as soon as the advice that changes the assignment is adapted by the agents. Second, the order is highly sensitive to noise tasks that are not part of the recurring tasks. The order can therefore change spontaneously without real impact on the recurring tasks. Third, exception rules that for instance let agents ignore tasks are not in all cases fit to establish a certain order. Exception rules that for instance boost tasks let the agents prefer the tasks that should be serviced before the others.

6.2.5.4 Rule Creation

To force the emergent system to behave in a similar way as the optimal solution in future, the sub-action $create \in Act_{EIA}$ instantiates based on the determined difference of the assignments a new exception rule (see Definition 6.6) for agent Ag_j^2 to change its behavior when it encounters a situation that was leading to the suboptimal behavior.

Definition 6.27 (Rule creation)
The creation of a rule is a function defined as

$$create : Assign \times Assign \rightarrow Rule$$

where

- $Assign$ is a set of assignments
- $Rule$ is a set of rules

The event and conditions specified by sit' respectively dat' of an exception rule r' are derived from the descriptions of the assigned tasks. For determining sit' and dat', Ag_{EIA} looks up the assignment $(sit, dat, act_{ta_j^2}) \in GHist_{Ag_j^2}$, which represents in the history of Ag_j^2 the point in time when it choses to perform $act_{ta_j^2}$. (sit', dat') is then an abstraction of sit and dat. This tuple is application dependent and tries to cover not only ta_j^2, but the whole cluster of tasks from the action $extract$ of which ta_j^2 is a member of. The types of exception rules that are able to change the behavior of an agent depends on the application domain and the basic MAS. However, the action of the exception rule has to change the sequence of actions an agent performs when the predicate for its current situation is $true$.

The exception rules created for an agent Ag_i are stored in the data structure $dat_{EIA}^{Rule_i}$. The possible rule set for this agent is denoted as $Dat_{EIA}^{Rule_i}$. Consequently, the possible rules sets for all agents in A with $m = |A|$ are denoted as

$$Dat_{EIA}^{Rule_A} \subseteq Dat_{EIA}^{Rule_1} \times Dat_{EIA}^{Rule_2} \times \ldots \times Dat_{EIA}^{Rule_m} \tag{6.5}$$

Based on the four sub-actions, the action $derive$ can be defined fully.

Definition 6.28 (Derive)
The derivation of rules and is a function defined as

$$derive : Dat_{EIA}^{GHist_A} \times Dat_{EIA}^{Tasks} \times Dat_{EIA}^{Sol_{opt}} \times Dat_{EIA}^{Sol_{emerg}} \rightarrow Dat_{EIA}^{Rule_A}$$

where

- $Dat_{EIA}^{GHist_A}$ is the set of all possible global histories of a set of agents A stored in Dat_{EIA}
- Dat_{EIA}^{Tasks} is the set of all possible sets of recurring tasks stored in Dat_{EIA}
- $Dat_{EIA}^{Sol_{opt}}$ is the set of all possible optimal solutions for a set of recurring tasks and a set of agents stored in Dat_{EIA}
- $Dat_{EIA}^{Sol_{emerg}}$ is the set of all possible emergent solutions for a set of recurring tasks and a set of agents stored in Dat_{EIA}
- $Dat_{EIA}^{Rule_A}$ is the set of all possible rule sets for a set of agents A stored in Dat_{EIA}

6.2.6 Send Rules to Agents

The main task of the action $send \in Act_{EIA}$ is to transfer a newly created exception rule for an agent Ag_i, the next time Ag_{EIA} can exchange information with this agent.

Definition 6.29 (Send)
The sending of a set of rules $Rule_i$ by the agent Ag_{EIA} to an agent Ag_i is a function defined as

$$send : Dat_{EIA}^{Rule_i} \rightarrow Dat_i^{Rule}$$

The agent will receive the exception rule and store it in its internal data structure dat_i^r. From this moment on, the rule will be incorporated into its decision mechanism as described in Definition 6.1.

6.2.7 Data Model

Based on the aforementioned actions and defined functions, the super-action $advise \in Act_{EIA}$ and the required data model Dat_{EIA} can now be defined.

Definition 6.30 (Advise)
The advise for a set of agents A by an agent Ag_{EIA} is a function defined as

$$advise : Hist_A \rightarrow Rule_A$$

Based on the received local agent histories, the action *advise* provides advises to the agents in A in form of exception rules. The data model Dat_{EIA} therefore contains all necessary values and intermediate results for the actions *receive*, $transform$, *extract*, *optimize*, *derive*, and *send*.

Definition 6.31 (Data model)
The data model of Ag_{EIA} is defined as

$$Dat_{EIA} \subseteq Dat_{EIA}^{Env} \times Dat_{EIA}^{Hist_A} \times Dat_{EIA}^{GHist_A} \times Dat_{EIA}^{Tasks} \times Dat_{EIA}^{Sol} \times Dat_{EIA}^{Rule_A}$$

where

- Dat_{EIA}^{Env} is the set of all possible representations of the environment of the system
- $Dat_{EIA}^{Hist_A}$ is the set of all possible local agent histories
- $Dat_{EIA}^{GHist_A}$ is the set of all possible global histories
- Dat_{EIA}^{Tasks} is the set of all possible recurring tasks
- Dat_{EIA}^{Sol} is the set of all possible (optimal/emergent) solutions for a set of recurring tasks
- $Dat_{EIA}^{Rule_A}$ is the set of possible rule sets for all agents

6.3 Realization Aspects

The generic advisor model described in the previous section can be realized and customized in different ways for various application domains, agent models, as well as coordination models (see for instance later Section 7.3). However, there are some aspects of the generic model that are of worth to think about before it is instantiated for a certain application:

- **Functional distribution:** As described in the last section, the advisor usually is not an element of the agent set of the basic self-organizing emergent MAS itself, but rather an additional element. Nonetheless, it is also conceivable that the advisor can be a role of one of the agents, or even all agents in the basic system can share performing the functions of the advisor. However, this requires extensive communication between the agents and might require more computing power in an agent than is possible in a particular application. A dedicated agent with lots of computing power and occasional communication with the agents is a more reasonable extension to many existing systems for the problems one is usually interested in.

- **Rule specification:** There are several ways to specify the situation of an exception rule's condition an agent can be in at runtime. For certain applications, the specification of concrete situations may be appropriate, which describe a situation in a very exact way and are therefore only valid if the situation is exactly as described. For other applications, prototypical situations may be more appropriate. As they are not concrete, a notion of similarity for such situations has to be established before. An agent then compares the perceived situation with the specified prototypical situation and if the similarity is higher than a certain threshold, it will perform the action of the exception rule. The way in which the action of exception rules is specified depends largely on the adaptability of the agents and the parameters that can be changed.

- **Rule lifetime:** To keep the local rule base of an agent clear and light, in particular for applications in which agents are equipped only with little memory or processing capacities, exception rules may be attributed with a certain time to live. Agents thus may "forget" exception rules of the advisor if they were not applied for a certain amount of time. This ensures that exception rules that are no longer applicable due to changed environmental constraints don't clutter the rule base.

- **Persistence of the data model:** For certain applications it is not necessary to store the data of the advisor to a database system or another persistent storage. The only important aspect is that the data of several run instances is available to the advisor. Nevertheless, for other applications it can be more appropriate to access historical data to facilitate analysis and simulation. If the data model is designed in a way that allows it to be stored, the data collected during the execution of a run instance can be stored persistently, so that it can be retrieved at a later point in time, e. g. for analysis or further optimization.

- **Correctness of data:** The described concept of an advisor assumes that the data collected by the agents, i. e. their local histories, is exact and correct. For certain applications, both assumptions may be loosened, depending on the reliability and trustworthiness of the agents respectively their sensors in the first instance. For applications with possibly unknown agents, the action $transform$ will have to take into account the uncertainty of each piece of information.

- **Usage of data:** The proposed approach of an advisor currently uses higher-level data in the form of global agent and system histories as well as tasks. This allows the advisor to adapt the assignment of agents to tasks and to influence the system based on this very high-level view. Depending on the application domain and concrete basic systems, the lower-level agent data might contain additional data that can be used in the optimization process. Even though this data is not used in the current model, the advisor already contains the

infrastructure necessary to create rules from all data that was collected by the agents. There is no limitation in how the data is used, processed, and applied to the agent system.

6.4 Related Work

The overall functionality of the EIA approach at a first glance has certain similarities to Model Predictive Control (MPC) [CB04]. In contrast to feedback control, which considers the controlled system as a kind of black box, MPC is a form of control that makes explicit use of a linear model of the controlled system. The model is used to predict the system output at future time instants (prediction horizon). The current control action is then obtained by minimizing an objective function, i. e. at each sampling instant a finite horizon open loop optimal control problem is solved, using the current state of the controlled system as the initial state. The optimization yields an optimal control sequence and the first control action in this sequence is applied to the controlled system. At each time instant the prediction horizon is displaced towards the future, which involves the application of the first control action calculated at each instant. This process is repeated whenever a new system state is available. For instance, in the area of AC, in [AK07] an MPC framework is presented that uses 'limited lookahead control' to optimize the forecast behavior of the controlled system over a limited prediction horizon. Based on a stochastic model, the controller within the framework predicts from the current state all possible (or at least a set of) future system states up to a certain prediction horizon and based on that choses the first control action of a sequence that optimizes the given constraints. This framework is applied to processor power management and distributed signal classification.

Similar to the EIA approach, MPC yields high performance control systems capable of operating without expert intervention for long periods of time. However, MPC is mainly used in the process industry and as a consequence the controlled system very often is considered to by a chemical or energetic process. In particular, the controlled system is *not* supposed to solve any dynamic optimization problem. Self-organizing emergent systems, by contrast, which in particular are supposed to solve dynamic optimization problems, are moreover characterized by non-linearity (see Subsection 2.3.7), which complicates the application of linear models. Even though Nonlinear Model Predictive Control (NMPC) explicitly takes account of nonlinear systems by the use of nonlinear models, the availability of nonlinear models (either from experimental data or formal theory) is still an open issue (cf. [CB04]), in particular for self-organizing emergent systems. Even if nonlinear models in general could be obtained via supervised learning, where the system is first simulated for various environmental inputs, the openness of self-organizing emergent systems operating in unknown situations prevents the accurate determination of such models for this class of systems. Consequently, the EIA cannot predict the future output (efficiency) of self-organizing emergent systems based on their current state using

MPC techniques. It rather learns respectively uses a model of the problem to solve and determines based on the solution of this problem by a static optimization algorithm control actions to be applied to the individual elements of the controlled system. A specific model of the controlled system is even not required, because the EIA only presumes that an agent participating in the self-organizing emergent solution is extended to deal with exception rules. As a consequence, the EIA does not consider the entire self-organizing emergent system as a back box but only the agents to advice.

In order to advice the agents, an EIA realizes a feedback control loop. As a consequence, the EIA approach is related to the approaches for the adaptation of self-organizing emergent systems listed in Section 5.4. Due to the very general description of the generic O/C framework [RMB+06], on an abstract level the functions of the EIA can be mapped to this framework as well. The functions *receive*, *transform*, and *extract* constitute the *observer* function, whereas the functions *optimize*, *derive*, and *send* constitute the *controller* function. However, when comparing the EIA approach to the existing instantiations of the O/C architecture, it becomes apparent that the EIA functionality exceeds their capabilities with regard to the general adaptation constraints (see Table 6.1). In particular, the EIA approach assumes that due to various communication constraints an EIA is able to observe the elements of the controlled system only at distinct points in time and distinct places in the environment, which respects the general characteristics of self-organizing emergent systems. Therefore, an EIA is able to adapt the local behavior of the system elements for *future* situations, instead of controlling and influencing the *current* system output. In contrast to O/C instantiations that adapt the local behavior of system elements as well, an EIA only gives advice to the system elements, but does not change their entire behavior or parameter set (as e. g. in [PRT+08]). An EIA respects the autonomy of the adapted system elements, as the advice can be ignored by the elements. Furthermore, the EIA approach considers the openness of self-organizing emergent systems with regard to unknown situations as well as a changing number of system elements. In contrast to most O/C instantiations as well as the MBE approach [SLT08], the EIA approach incorporates online learning to identify the reccuring task of the problem.

Due to the very general description of the reference model of an AM [IBM06] in the area of AC, on an abstract level the functions of the EIA can be mapped to this framework as well. The function *receive* can be mapped to the function *monitor*, the functions *transform* and *extract* can be mapped to the function *analyze*, the functions *optimize* and *derive* can be mapped to the function *plan*, and the function *send* can be mapped to the function *act*. However, similar to all other adaptation approaches, an AM assumes to have a high observability and good controllability of the controlled system. Although an AM might be used to adapt the behavior of a managed element, in contrast to an EIA, an AM – due to its business background – is intended to perform a very strong regulation and thus limits the autonomy of the managed elements to a minimal level.

On a very abstract level, a big difference to most other approaches lies in the

	O/C [MRB+07]	O/C [PRT+08]	O/C [CHMS08]	O/C [RRS08]	O/C [BAU+10]	MBE [SLT08]	EIA
Regard for low observability							+
Regard for poor controllability	(+)						+
Adaptation of local element behavior		+				+	+
Preservation of basic behavior	+			+			+
Focus on system performance		(+)	+		+	+	+
Consideration of openness	+		+	(+)	+	+	+
Consideration of autonomy	+	(+)		+	+	+	+
Incorporation of online learning		+			+		+
Incorporation of offline learning		+					

Table 6.1: Comparison of approaches for adapting self-organizing emergent systems

level of operation. In [NOR03], three levels of behavior in animals and humans are described and then transferred to artificial systems: *reaction*, which deals with predefined, undeliberated responses to sensory input (see e. g. [CHMS08, RRS08, SLT08]); *routine*, the level on which learned behavior is executed and the consequences of actions are assessed (see e. g. [PRT+08, BAU+10]); and *reflection*, the level on which a system deliberates about itself, its past, and future behavior. According to this categorization, the EIA operates on the third level (reflection), as it analyzes the actions of the system and adapts its constituent parts to increase the system's efficiency. However, to be really able to compare self-adaptation approaches on an abstract level more thoroughly, some kind of commonly agreed formalization of the respective functionalities is required, as e. g. suggested and started in [WMA10].

6.5 Conclusion

In this chapter we have presented the EIA approach, which is able to adapt at runtime the local behavior of elements (agents) in self-organizing emergent systems solving dynamic optimization problems with recurring tasks. Due to these adaptations, the EIA approach facilitates a more efficient operation of self-organizing emergent systems (see *Objective 2*).

In more detail, the EIA represents an approach that implements the principles of self-adaptation in order to compensate for the runtime insufficiencies of self-organizing emergent systems. Thereby, the approach considers the general constraints for the adaptation of self-organizing emergent systems, which result from their specific system characteristics compared to conventional computer systems (see

Section 5.4). In particular, the EIA approach takes into account the low observability and poor controllability of self-organizing emergent systems (see *Challenge 4*), which – in comparison to existing approaches – facilitates the adaptation of self-organizing emergent systems in a wider range of application domains, in which communication may be very costly, locally forbidden, globally restricted, structurally infeasible, or only temporally possible. The adaptation of the individual local behaviors of system elements is accomplished by providing the elements with advice, how to behave more optimally in predicted future situations. The advice is provided in the form of exception rules, which are derived from a calculated optimal solution. All advices preserve the basic self-organizing and emergent behavior of the systems as well as their beneficial properties scalability, robustness, flexibility, and adaptivity (see *Challenge 5*). Moreover, all problem-solving decisions are still made by the system elements themselves. Thereby, the advices can be ignored by the elements and in particular will be ignored, if they have not been applied for a certain amount of time, e. g. if a predicted situation did not occur in future. This is a tribute to the openness and autonomy of these self-organizing emergent systems (see *Challenge 6*).

The advices provided by an EIA, i. e. the exception rules, help the system elements to make more optimal local decisions with regard to the global efficiency of the produced solution. Dependent on the application domain and the controlled basic MAS, the exception rules are able to change the local behavior of an agent respectively the sequence of actions an agent performs when the predicate of the exception rule for the current situation is true. This compensates for the runtime insufficiencies of self-organizing emergent MASs (see Section 5.1). In particular, the reactiveness and greediness of agents can thus be influenced very effectively. Moreover, the exception rules provide the agents with a limited capability to 'look into the future', such that a dynamically appearing task is assigned to the best agent with respect to the global optimality of the solution. The experiments described in Section 8.4 will demonstrate these capabilities by resilient results for a concrete application domain. The next chapter introduces this application domain and provides a customized instantiation of the EIA approach for this domain.

Part IV

Applications and Experiments

Chapter 7

Application Domain: Pickup and Delivery Problems

While the chapters of the first and the second part of this thesis dealt with the design respectively the operation phase of self-organizing emergent systems in theory, the chapters of the third part of this thesis deal with a concrete application of the developed concepts to a problem domain including its experimental evaluation. More specifically, this chapter presents the instantiations of the IBC approach (see Chapter 4) and of the EIA approach (see Chapter 6) as well as their application to the problem domain of vehicle routing [CLSV07], in more detail to the Pickup and Delivery Problem (PDP) [DDE+02].

Roughly spoken, the PDP concerns the service of a set of customers in a given time period by a set of vehicles, which are located in one or more depots and perform their movements by using an appropriate road network (even though PDPs can be extended to air, water, and rail networks as well). A solution of a PDP calls for the determination of a set of routes, each performed by a single vehicle that starts and ends at its own depot, such that all requirements of the customers are fulfilled, all the operational constraints are satisfied, and one or more global optimization objectives are reached (cf. [TV02]). Practically, the PDP is an omni-present problem that is faced each day by thousands of companies and organizations engaged in the delivery and collection of goods or people. It appears in various domains, such as courier services, manufacturing control, aircraft sharing, dial-a-ride transportation, container terminals, distribution of heating oil, taxi cab services, emergency vehicle dispatching, or even patient transportation in hospitals, to name only a few. For example, the world's largest package delivery company, UPS, every day transports 15.1 million packages and documents worldwide from 1.8 million pick-up customers to 6.1 million delivery customers by almost 100.000 package cars, vans, tractors, motorcycles and more than 500 aircrafts [UPS10]. Apparently, such companies continuously seek to improve their transportation network performance in order to tap the full potential of possible OPEX reduction, which requires flexible and efficient solutions able to cope with high dynamics.

The remainder of this chapter is thus organized as follows: Section 7.1 provides the background and terminology of vehicle routing and pickup and delivery problems, including an appropriate formal problem definition and existing solution methods. Section 7.2 presents the instantiation of the IBC approach for the solution to PDPs,

more specifically a flexible and efficient, decentralized coordination mechanism utilizing a biological paradigm. Subsequently, Section 7.3 presents the instantiation of the EIA approach for the efficiency improvement of the IBC-based solution. Based on these instantiations, Section 7.4 discusses several aspects, how to extend these instantiations in future in order to improve the overall efficiency further. Finally, Section 7.5 concludes this chapter.

7.1 Pickup and Delivery Problems

The PDP, which is sometimes also referred to as Transportation on Demand (TOD) [CLPS07], is a generalization of the VRP [TV02, GRW08], which is itself a generalization of the well-known Traveling Salesman Problem (TSP). Hence, PDPs are in general NP-hard [GJ79]. Research in the field of vehicle routing, whether involving pickups and deliveries or not, originally started with the so-called truck dispatching problem introduced by Dantzig and Ramser [DR59] in 1959. The objective was to find the *"optimum routing of a fleet of gasoline delivery trucks between a bulk terminal and a large number of service stations supplied by the terminal"*. The solution at that time consisted of 4 vehicles operating 12 routes. Over the past 50 years, extensive research, mainly in the fields of Operations Research (OR) and Mathematical Programming, has then been dedicated to the domain of vehicle routing. In particular vehicle routing involving both, pickups and deliveries, has received considerable attention and the abundance of research results have paved the way for many practical real-world applications, as demonstrated by the UPS example.

7.1.1 Problem Classification

The long time of research in this field, however, has also led to a somewhat confusing terminology used to describe the various problem types arising in this context. Because an exhaustive overview on all terms used is out of the scope of this thesis, we only present some widespread classification schemes that illustrate the various types of problems.

Savelsbergh and Sol [SS95] provide an abstract classification scheme by the formal definition of the General Pickup and Delivery Problem (GPDP). This was the first attempt to provide an unified notation for all versions of the PDP. In the GPDP, a set of routes for a fleet of vehicles has to be constructed in order to satisfy transportation requests. A transportation request is specified by the size of the load to be transported, the locations where it is to be picked up (the origins) and the locations where it is to be delivered (the destinations). Each load has to be transported by one vehicle from its set of origins to its set of destinations without any transshipment at other locations. Each vehicle has a given capacity, a start location, and an end location. The authors subdivide the GPDP into three main classes: the classical VRP, in which either all the origins or all the destinations are located at the depot, the classical PDP, in which each transportation request specifies a single origin and a single destination and all vehicles depart from and return to a central

depot, and the Dial-A-Ride Problem (DARP) [CL03, CL07], in which the loads to be transported represent people (then called clients). The difference between PDPs and DARPs is usually expressed in terms of additional constraints or objectives that explicitly take client (in)convenience into account.

Berbeglia et al. [BCGL07] use a more fine-grained three-field classification scheme for pickup and delivery problems, which is [*structure*|*visits*|*vehicles*]. The first field (*structure*) specifies the number of origins and destinations of the goods to be transported. In so-called *many-to-many* (M-M) problems, any customer can serve as a source or as a destination for any good. According to the authors, many-to-many problems are, however, not frequently encountered in practice. In so-called *one-to-many-to-one* (1-M-1) problems, the goods are initially available at the depot and are transported to the customer locations [GL08]. In return, goods available at the customers are transported to the depot. Finally, in so-called *one-to-one* (1-1) problems, each good has a given origin and a given destination [CLR08]. The second field (*visits*) provides information on the way pickup and delivery operations are performed at customer locations, i. e. either each customer is visited exactly once for a combined pickup and delivery operation, denoted as *PD*, the two operations may be performed together or separately, denoted as *P-D*, or either a pickup or a delivery operation is performed at each customer, but not both, denoted as *P/D*. The third field (*vehicles*) provides the number of vehicles used in the solution to the problem. The entry '*1*' stands for a single-vehicle problem, '*m*' for multi-vehicle problem, and '–' for an undefined number of vehicles.

Parragh et al. [PDH08a, PDH08b] provide a more detailed classification scheme in order to clarify the various problem types. Figure 7.1 depicts a mapping of this classification scheme to the two aforementioned schemes. Basically, they subdivide pickup and delivery problems into two main problem classes, the Vehicle Routing Problem with Backhauls (VRPB) and the Vehicle Routing Problem with Pickups and Deliveries (VRPPD). In VRPBs, all goods delivered have to be loaded at one or several depots, and all goods picked up have to be transported to one or several depots. Thereby, the authors denote delivery customers as linehaul customers and pickup customers as backhaul customers. The VRPB is then further subdivided into four minor subclasses. In the first two subclasses, customers are either delivery or pickup customers but never both. In the last two subclasses, each customer requires a delivery and a pickup. In the Vehicle Routing Problem with Clustered Backhauls (VRPCB) all linehauls have to be served before the backhauls. The Vehicle Routing Problem with Mixed linehauls and Backhauls (VRPMB) permits, by contrast, any sequence of linehauls and backhauls. In the Vehicle Routing Problem with Divisible Delivery and Pickup (VRPDDP) customers demanding delivery and pickup service can be visited twice, whereas in the Vehicle Routing Problem with Simultaneous Delivery and Pickup (VRPSDP) customers demanding both services have to be visited exactly once. By contrast, in the second major subclass, the VRPPD, goods respectively passengers have to be transported between pickup and delivery customers respectively points. The VRPPD is then also further subdivided into two minor subclasses. The first subclass refers to problems where pickup and

delivery locations are unpaired. Therefore, a homogeneous good is considered so that each load picked up can be used to fulfill the demand of any delivery customer. This problem class is denoted as Pickup and Delivery Vehicle Routing Problem (PDVRP). The second VRPPD subclass comprises the classical PDP and the classical DARP. Both classes consider transportation requests, each associated with an origin and a destination, resulting in paired pickup and delivery points.

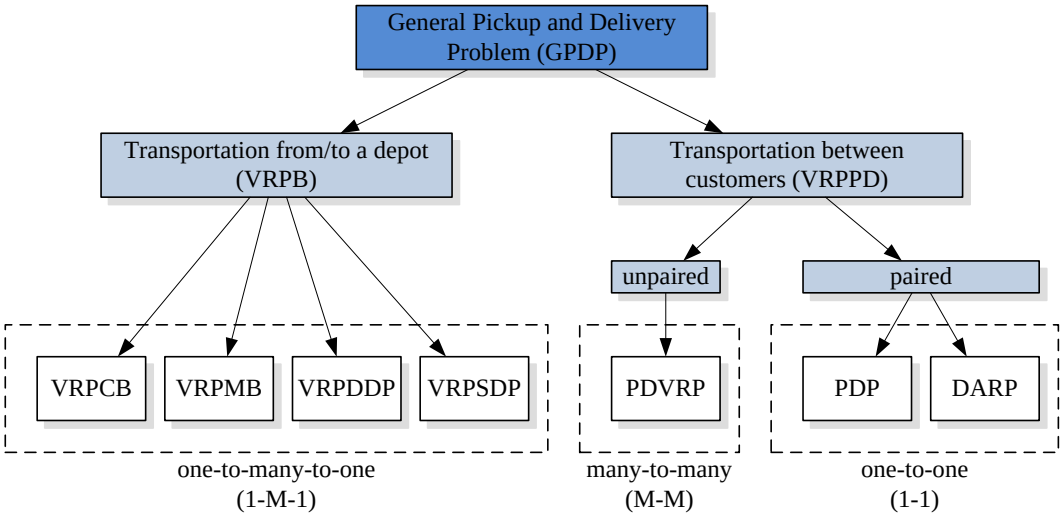


Figure 7.1: Different classification schemes of pickup and delivery problems

Without doubt, every of these three mentioned classification schemes helps to clarify the terminology of pickup and delivery problems. However, most problem classes additionally have a couple of further dimensions. Almost all classes can be distinguished between *full-truck-load problems* and *less-than-truck-load problems*. Full-truck-load problems are a special case in which each good has to be transported directly from its origin to its destination. Less-than-truck-load problems may in contrast also comprise other pickup or delivery stops between the fulfillment of a specific request. Another dimension considers the number of vehicles. This leads to a differentiation between *single-vehicle-problems* and *multi-vehicle-problems* (see also the classification scheme of Berbeglia et al.). A further dimension regards time constraints. In a pickup and delivery problem with *time windows* (PDPTW), a time interval is associated with each customer [DDS91, CDSS02]. Also, the time step in which the vehicles leave the depot, a travel time, and an additional service time for each customer are given. The service of each customer must then start within the associated time window and the vehicle must stop at the customer for the respective service time. In case of early arrival at the location of the customer, the vehicle is allowed to wait until the time window opens up, i. e. until the service may start.

In general, pickup and delivery problems can also be differentiated according to the availability of information. In *static* problems (see e. g. [BCGL07]), all information relevant to a problem (usually the transportation requests, travel times, etc.) is

known before the routes for the vehicles are constructed. This is a realistic assumption in contexts where users specify transportation requests one or two days in advance, as it is often the case in a DARP. However, more often the arrival time of new transportation request, the location of the new requests, and the travel time between customers are not known a priori, or other unexpected events occur, such as vehicle breakdowns, crashes, traffic congestions, withdrawn transport requests, drivers calling sick, early arrivals, and imprecise load sizes. Thus, in *dynamic* problems (see e. g. [Psa88, GP98, BCL10]), some of the relevant information is revealed or may be updated during the period of time in which operations take place. Consequently, in a dynamic problem, when a new transportation request becomes available, at least one route has to be changed in order to serve this new request. In contrast to a static problem, the planning horizon of a dynamic problem may be unbounded. However, even in dynamic problems some transportation requests may be already available at the start of planning. By contrast, in *stochastic* problems (see e. g. [GLS96]), some relevant information is also present a priori, whereas some relevant information is however only random variables, whose distributions are usually known.

7.1.2 Problem Definition

In order to define the PDP more formally, we reuse existing definitions from the OR community (see [DDS91, DDE+02, SS95, BCGL07, BCL10, CLPS07, FHK+07, PDH08a, PDH08b]). In a PDP, usually z pickup customers (pickup stations) and $\tilde{z}$ delivery customers (delivery stations) together form a set of transportation requests R that are to be served by a fleet of y vehicles. Because the PDP (as well as the DARP) consider settings in which pickup and delivery stations are paired, $z = \tilde{z}$. Thus, the set of pickup stations is denoted as $PS = \{s_1, \ldots, s_z\}$ and the set of delivery stations is denoted as $DS = \{s_{z+1}, \ldots, s_{z+\tilde{z}}\}$. It is however possible that different stations represent the same geographical location. Consequently, $S = PS \cup DS$ is the set of stations where goods can be picked up or be delivered to, while the set $S_0 = S \cup \{0\}$ additionally includes a depot. A depot is characterized by the number and types of vehicles associated with it. A vehicle v is characterized at least by its initial location, which is usually the depot, its capacity *cap* (expressed as the maximum weight, volume, or number of packages the vehicle can load), and possibly its type or compartment *comp* (characterized by the type of goods that can be carried). The set of vehicles is denoted as $V = \{v_1, \ldots, v_y\}$. Sometimes, the s^{th} pickup station is denoted by s^+ and the associated delivery station by s^-. q_s denotes the demand respectively supply of a station s. Pickup stations are thereby associated with a positive value, delivery stations with a negative value. At the depot the demand/supply is zero. Moreover, st_s denotes the service time at station s, i. e. the time required to collect or deliver goods at the station.

S_0 is placed on a map, which is represented as a graph $M = (L, C)$ that contains a set of locations L, which correspond to junctions, customer locations, the depot, or other relevant points, i. e. $S_0 \subseteq L$, as well as a set of connections $C = \{(l_i, l_j) \mid l_i, l_j \in L, i \neq j\}$ that connect these locations. For the purpose of this thesis, M is defined

as a directed graph. d_{ij} denotes the distance from location i to location j, t_{ij}^v the travel time for vehicle v from i to j, and c_{ij}^v the nonnegative travel costs[1][2] for vehicle v from i to j[3]. In a PDPTW, the interval $[t_s^{start}, t_s^{end}]$ additionally specifies the time window on a pickup or delivery station s. The parameter t_s^{start} is called the release time of station s, while t_s^{end} is called the deadline, where $t_s^{start}, t_s^{end} \in Time, \forall s \in S_0$. Furthermore, some decision variables are required:

- $x_{ij}^v = \begin{cases} 1, & \text{if vehicle } v \text{ travels from location } i \text{ to location } j \\ 0, & \text{else} \end{cases}$
- Q_s^v is the load of vehicle v when leaving station s
- $t_s^{begin,v}$ is the beginning of the service of vehicle v at station s

For the purpose of this thesis, we can now define a transportation request as a task in a PDP in the sense of Definition 6.1.

Definition 7.1 (Transportation request)
A transportation request is defined as a task in a PDP ta^{PDP} such that

$$ta^{PDP} = (s_1, s_2, q, t_{s_1}^{start}, t_{s_1}^{end}, t_{s_2}^{start}, t_{s_2}^{end})$$

where

- $s_1 \in PS$ is the pickup station
- $s_2 \in DS$ is the delivery station
- q is the supply of s_1 respectively demand of s_2, generally called loadsize
- $t_{s_1}^{start}, t_{s_2}^{start} \in Time$ are the release times of s_1 respectively s_2
- $t_{s_1}^{end}, t_{s_2}^{end} \in Time$ are the deadlines of s_1 respectively s_2

The definition of a task now allows the definition of PDPs for the purpose of this thesis.

Definition 7.2 (Pickup and Delivery Problem)
A Pickup and Delivery Problem is defined as a quadruple

$$PDP = (M, S_0, R, V)$$

[1] The cost is defined as the Eucledian distance between the two locations and therefore satisfies the triangle inequality: $c_{ik}^v + c_{kj}^v \geq c_{ij}^v, \forall i, j, k \in L, \forall v \in V$.

[2] The costs are usually associated with the distance or the travel time between the two locations.

[3] The literature in the field of OR often transforms the original road graph into a complete graph (see e. g. [SS95, TV02, PDH08b]), whose vertices are the vertices of the road graph corresponding to the customers and the depot. The costs between vertices i and j are then given by the shortest path starting from vertex i and arriving at vertex j. The travel time is computed as the sum of the travel times of the connections belonging to the shortest path. For the purpose of this thesis, however, we use the original road graph for the definition.

where

- M is the environment map
- S_0 is the set of pickup and delivery stations including a depot
- R is the set of transportation requests
- V is the set of vehicles

A solution sol^{PDP} (see Definition 6.4) to a PDP then consists of assignments (see Definition 6.3) of the vehicles in V to all transportation requests in R such that

1. every vehicle starts a route at the depot and returns to the depot at the end of its route:

$$\sum_{j:(0,j)\in C} x^v_{0j} = 1, \forall v \in V \tag{7.1}$$

$$\sum_{i:(i,0)\in C} x^v_{i0} = 1, \forall v \in V \tag{7.2}$$

2. every vehicle's capacity is not exceeded throughout its tour:

$$x^v_{s_1 s_2} = 1 \Rightarrow Q^v_{s_2} = Q^v_{s_1} + q_{s_2}, \forall s_1, s_2 \in S_0, \forall v \in V \tag{7.3}$$

$$max\{0, q_s\} \leq Q^v_s \leq min\{cap^v, cap^v + q_s\}, \forall s \in S_0, \forall v \in V \tag{7.4}$$

3. all time constraints are satisfied:

$$x^v_{s_1 s_2} = 1 \Rightarrow t^{begin,v}_{s_2} \geq (t^{begin,v}_{s_1} + st^v_{s_1} + t^v_{s_1 s_2}) x^v_{s_1 s_2}, \forall s_1, s_2 \in S, \forall v \in V \tag{7.5}$$

4. a pickup and its associated delivery are served by the same vehicle (pairing constraint):

$$\sum_{j:(s^+,j)\in C} x^v_{s^+ j} - \sum_{j:(s^-,j)\in C} x^v_{s^- j} = 0, \forall s \in S, v \in V \tag{7.6}$$

5. a pickup is always made before its associated delivery (precedence constraint):

$$t^{begin,v}_{s^+} \leq t^{begin,v}_{s^-}, \forall s \in S, v \in V \tag{7.7}$$

6. one or more objectives are fulfilled

Because conditions vary from one setting to the next, different objectives, ordered hierarchical or equal, can be found in literature. In most cases, the total travel costs have to be minimized:

$$min \sum_{v \in V} \sum_{(i,j) \in C} c_{ij}^{v} x_{ij}^{v} \tag{7.8}$$

Other objectives that can be found are to minimize the travel time of each vehicle, the number of vehicles required, the duration of a route, or the total completion time. For some settings, vehicles can even operate more than one route in the considered time period. For instances of the PDPTW in addition to the above constraints all time windows have to be satisfied:

$$t_s^{start} \leq t_s^{begin,v} \leq t_s^{end}, \forall s \in S, v \in V \tag{7.9}$$

Thereby, the deadline can be a hard or a soft constraint. In the latter case, a vehicle is allowed to arrive late, but a penalty is incurred in the objective. If a vehicle arrives too early, it has to wait without any penalty (neglecting any other costs such as driver costs or opportunity costs). Furthermore, a vehicle is allowed to wait at its initial location.

In OR literature, the solution to the PDP often includes further constraints, which simplify the calculation of a solution but are, however, not used in this thesis due to different reasons:

- **Each station is served exactly once** respectively **each transportation request is assigned to one vehicle only:** Even though this constraint may be useful for certain PDP settings, e. g. longhaul courier services, for other settings it is not. For example, consider a situation in which a pickup station continuously produces a certain type of good. If two vehicles, each loaded only half full, pass by this station, no one would be allowed to serve this station and a third vehicle might be required or one of the two vehicles would have to return to this station. Without this constraint, the solution becomes more flexible, as each vehicle possibly could load half of the goods, which saves time and costs.

- **All vehicles leave the depot at the point in time 0:** This constraint is only useful for static PDP(TW)s, where all requests are already known at time step 0. In dynamic versions of these problems, this constraint is not useful.

- **Routes may not contain subtours:** This constraint requires that all routes are connected to the depot and no tours that are not connected to the depot occur during a route. Again, this constraint may be useful for static PDP(TW)s, but dynamic and flexible solutions require to react immediately to new requests, without returning to the depot every time.

Assigning transportation requests to vehicles in the PDP is in general much more difficult than assigning transportation requests to vehicles in the VRP. In the VRP, all the origins of transportation requests are located at the depot. Therefore, transportation requests with geographically close destinations are likely to be served by the same vehicle. In the PDP, geographically close destinations may have origins that are geographically far apart. Thus, it is hard to conclude that they are likely to be served by the same vehicle.

The measurement of the performance or quality *qual* of a solution (see Definition 6.5) to dynamic PDPs, in which we are interested in, is complicated due to the different shapes of these problems. In order to better distinguish dynamic PDPs, Lund et al. [LMR96] have defined a *degree of dynamism* (dod) as the ratio of the number of dynamic requests $|R_{dyn}|$ to the number of total requests $|R|$, i. e. the sum of static requests $|R_{stat}|$ and dynamic requests:

$$dod = \frac{|R_{dyn}|}{|R|} = \frac{|R_{dyn}|}{|R_{stat}| + |R_{dyn}|} \tag{7.10}$$

where $0 \leq dod \leq 1$. Apparently, $dod = 0$ indicates a pure static PDP, whereas $dod = 1$ indicates a pure dynamic PDP. However, the performance of a solution to dynamic PDPs is assumed to be dependent not only on the number of dynamic transportation requests, but also on the time t^{avail} when these requests actually become available. Because the absolute measure proposed by Lund does not take the availability times of the dynamic transportation requests into account, Larsen [Lar01] extended this measure to the *effective degree of dynamism* (edod):

$$edod = \frac{\sum_{s=1}^{R_{dyn}} (\frac{t_s^{avail}}{Time})}{|R|} \tag{7.11}$$

where $0 \leq edod \leq 1$. The *edod* represents an average of how late the requests become available compared to the latest possible time the requests could become available.

In PDPTWs, a further important issue is the reaction time. It is defined as the temporal distance between the time a request becomes available to the system and the latest possible time (deadline) at which the service of the request should begin at a station s, i. e. $t_s^{end} - t_s^{avail}$. According to [Lar01], the effective degree of dynamism measure can then be extended to:

$$edod\text{-}tw = \frac{1}{|R|} \sum_{s=1}^{R} (\frac{Time - (t_s^{end} - t_s^{avail})}{Time}) \tag{7.12}$$

Apparently, the shorter the reaction time, the more dynamic is the PDPTW. The actual reaction time is, by contrast, the time a customer has to wait until his request becomes served, i. e. the waiting time. In general, minimizing the waiting time of customers is sometimes an objective of the solution to a PDP as well, but is usually more common to DARPs. The lower the waiting time, the higher is the degree of

service. However, the higher the effective degree of dynamism, the harder it is to achieve shorter waiting times.

7.1.3 Existing Solution Methods

The valuation of the nature of problems that can be encountered in practice has changed over the years. Whereas in 1959, Dantzig and Ramser [DR59] took static VRPs as a starting point, in 1995, Savelsbergh and Sol [SS95] stated that while most VRPs are static, most PDPs are dynamic. Psaraftis [Psa95], by contrast, argued in the same year that most real-world VRPs are dynamic, too. Powell et al. [PJO95], however, already recognized in 1995 that dynamic and stochastic problems will undoubtedly represent the *"wave of the future"*. Today, it is widely accepted that both the VRP and the PDP are by nature stochastic problems, but are in practice generally dynamic problems (cf. [FHK+07]), as significant relevant information will typically emerge as the routing plan is being executed, which may render a current routing plan infeasible or sub-optimal.

Consequently, during the last 50 years, plenty of solution methods to the various types of the PDP have emerged, emanating mainly from fields like OR and in recent years also MASs (respectively DAI). The primary difference between the methods of these fields is the level of control: centralized or decentralized. Whereas the focus of OR is due to its long history rather on centralized solution methods, the focus of research in MASs is on decentralized solution methods in the first instance. The question of which level of control is more appropriate is, however, not easy to answer. Thus, because both approaches have their advantages, in recent years more and more hybrid solution methods were proposed that combine both centralized and decentralized approaches to achieve better solutions. In the following, we give a brief overview on all three classes of solution methods, starting with classical, purely centralized solution methods (Subsection 7.1.3.1), proceeding via purely decentralized solution methods (Subsection 7.1.3.2), to hybrid solution methods (Subsection 7.1.3.3) at the end, along with their advantages and disadvantages. This forms a basis for the classification of the solution methods grounded on the IBC approach (see Section 7.2) and the EIA approach (see Section 7.3).

7.1.3.1 Centralized Solution Methods

Centralized solution methods can be physically embodied as a person or be virtually present due to high-levels of data aggregation in a central database. Recent overviews on centralized solution methods for *static* PDPs, i. e. the classical OR techniques, can e. g. be found in [BCGL07, PDH08b, CLR08]. Commonly, these solution methods are divided into three main classes: exact methods, heuristics, and metaheuristics.

Exact methods are divided into (1) tree-search methods, like branch-and-bound, branch-and-cut, branch-and-price, or branch-and-bound-and-price algorithms, (2) dynamic programming, and (3) integer-programming-based methods, such as col-

umn generation methods. Although exact methods produce optimal solutions, they are usually computationally expensive, even for relative small problem instances. The largest static PDP problem instance solved to optimality within a state-of-the-art exact method comprises 205 transportation requests [SPS04]. However, this instance was an only tightly constrained problem. Evidently, the problem size is not always a good indicator for the quality of a solution method. Today's well-balanced exact methods for static PDPs usually are only able to handle up to 75 transportation requests (cf. [PDH08b]).

In contrast, heuristics can solve problems with larger scales in less computation times than exact methods. However, heuristics usually lack robustness and their performance is very much problem-dependent. For instance, Fisher [Fis95] states that *"it's not uncommon that a heuristic developed for a particular geographic region of a company's operation will perform poorly in another region served by the same company"*. Commonly, solution heuristics are divided into (1) construction heuristics, (2) improvement heuristics, and (3) hybrid heuristics. Construction heuristics are further divided into pure constructions heuristics, such as insertion or savings heuristics, and two-phase heuristics, such as cluster-first-route-second or route-first-cluster-second heuristics. Hybrid heuristics integrate fast heuristics into the optimization framework of exact methods, in order to provide solution methods that are as robust as exact methods but also are capable of finding good solutions within acceptable computation time. For example, the largest static PDP instance solved by a heuristic within acceptable computational time comprises up to 500 requests [XCRA03]. The solution method was a hybrid heuristic based on column generation.

In contrast to heuristics, metaheuristics define abstract steps that at least theoretically can be applied to arbitrary application domains. Thereby, most metaheuristics are based on principles of physical or biological processes (see e. g. [Yan08]). Probably the most popular examples for metaheuristics in the PDP domain are *Simulated Annealing* and *Tabu Search*. Other metaheuristics include evolutionary algorithms, in particular genetic algorithms, neural networks, ant colony optimization [DS04], particle swarm optimization, or artificial immune systems, for instance, or hybrid mixtures of them as well. The solvable problem instance size is comparable to heuristics up to 500 transportation requests within acceptable time.

Whereas there is no commonly agreed benchmark data set to assess the performance of heuristics and metaheuristics for static PDPs, the benchmark data set most often used in literature for static PDPTWs is the one described in [LL01, LL02], which is based on the wide-spread benchmark instances proposed in [Sol87]. According to [PDH08b], the best results for these benchmarks have been presented in Ropke and Pisinger [RP06] and Bent and van Hentenryck [BH06], two metaheuristic solution methods.

In contrast to all these centralized solutions methods for static PDPs, the result of centralized solution methods to *dynamic* PDPs obviously cannot be a static output in the form of a set of routes. It rather entails devising a solution strategy that will adjust a current solution in the light of new relevant information [MML04] respectively a policy that prescribes how the routes should evolve as a function of those

inputs that evolve in real-time [Psa88]. In other words, the solution strategy uses the revealed information and specifies, which actions must be performed as time goes by. A basic and commonly used strategy for solving a dynamic PDP is to adapt an algorithm that solves the static version of the problem (cf. [BCL10]). Overviews on centralized solution methods to dynamic as well as stochastic PDPs can be found in [Psa88, Psa95, GGLM03, PDH08b, BCL10]. Two approaches can be distinguished. The first one consists of solving a static problem each time new information (such as a new transportation request or a cancellation) is revealed. One important drawback of this approach is that performing a complete re-optimization every time new information is revealed is too time consuming, and therefore inadequate for real-time settings where a fast response is required. In the second approach, which is the one generally used, a (relatively simple) static solution approach is applied only once at the beginning of the planning horizon to obtain an initial solution with the available information. When new information is revealed, the current solution is updated with heuristic methods such as insertion heuristics, deletion heuristics, or interchange moves, sometimes coupled with a local search algorithm (cf. [GGLM03]). In the intervals elapsed between the time instants at which new information is revealed, some more robust optimization methods are sometimes applied to improve the current solution.

Heuristics for (multi-vehicle) dynamic PDPs have been proposed amongst others by Savelsbergh and Sol [SS98], Popken [Pop06], as well as Fabri and Recht [FR06]. Metaheuristics have been proposed amongst others by Shen [SPRR95], Potvin [PSD95], Gutenschwager [GNV04], Mitrović-Minić et al. [MML04, MMKL04], Branke [BMND05], Montemanni [MGRD05], Pankratz [Pan05], Gendreau [GGPS06], Sáez [SCN08], and Ghiani [GMQT09]. Bio-inspired metaheuristics for static as well as dynamic PDPs have also been reviewed in [PT09]. In some cases up to 1000 transportation requests were considered in these heuristics and metaheuristics respectively, however, always with the option that some requests may be refused if they could not been handled within an acceptable time. Unfortunately, the heuristics and metaheuristics cannot be directly compared since no standardized simulation environment has been used in literature yet. Exact methods have not been used to solve dynamic or stochastic PDPs, so far.

Apparently, the biggest advantage of centralized solution methods is their production of an optimal or nearly optimal solution. It is commonly agreed that in settings where all or even most of the information is known in advance, i. e. static PDPs, OR techniques outperform agent-based approaches. Moreover, a centralized view increases the understanding of the problem to be solved and can be used to offer transparency in decisions. However, centralized solution methods entail a couple of disadvantages (cf. [Mvv07, DPH07]):

- In most cases they require a lot of information in advance.
- They can be very sensitive to information updates, i. e. a minor modification in information may have impact on the routes of many vehicles.

- The time required for the planning of the solution may not permit timely response to unexpected events such as vehicle breakdowns or the arrival of rush orders.
- They are not applicable to all problems because customers may not be willing to share all their critical information such as their cost structure, current vehicle locations, or current schedules.
- They are not always able to work with the real problem, i. e. many constraints that are soft in nature, are modeled as being hard constraints, or cannot be modeled at all. As a consequence, the modeling is either imprecise or makes wrong assumptions.
- They will fail (at least today) in very complex (high number of transportation requests) or/and dynamic (high degree of uncertainty about future transportation requests) problems
- They do not provide good scalability, flexibility, robustness, and adaptivity.

7.1.3.2 Decentralized Solution Methods

Modeling and solving problems by a set of coordinating agents implies a number of advantages that can overcome the aforementioned disadvantages of classical OR techniques. Such decentralized solution methods can be physically embodied as well, e. g. in vehicles or robots making decisions in the field, or virtually embodied in terms of multiple software components operating autonomously on the same server. Decentralized, agent-based solution methods in general are able to handle complexity and dynamism better than centralized approaches, in particular in settings where information becomes available at a very late timing. Whereas OR techniques may need too much time to re-optimize a solution when a sudden change occurs, agent-based solutions can be very reactive to such new events. Mahr et al. [MSdWdW08] have shown that agent-based approaches outperform OR techniques for settings in which less than 50% of the transportation requests are known in advance. Further advantages are the possibility for distributed computation, the ability to react quickly on local information, and the ability to deal with proprietary data from multiple companies, i. e. a MAS can support quick decision-making by producing a feasible solution to the problem at hand without revealing critical internal information of a company at any moment in time (cf. [FMPS95, BFV00]).

A key issue, however, again is how to design and configure agents in an easy, timely, and inexpensive manner such that their local, possibly self-interested behavior yields a near-optimal global solution to a PDP(TW) (cf. *Problem 1* at page 6). Thus, existing solution methods are based on models and mechanisms for decentralized coordination (see Subsection 3.3). A large part of solution approaches is grounded on market-based coordination, using auctions or negotiations (see Subsection 3.3.1). For instance, Boucke et al. [BWHM04] propose an extension of the

CNP by a negotiation protocol for flexible and decentralized allocation of tasks in dynamic PDPs, where the agents continuously reconsider the situation in the environment and adapt the assignment of tasks when circumstances change, in order to handle delayed commencement of tasks. Similarly, [ZLL09] propose another extension of the CNP that tries to reduce the number of messages required to solve a static PDPTW. Mes et al. [MvdHvH08] propose a decentralized solution to dynamic PDPs in an industrial bakery based on auctions, where automated guided vehicles (AGVs) are used in the dough making process.

Farinelli et al. [FINZ05] present an approach grounded on token-based coordination (see Subsection 3.3.4). They describe an example from robotics, in which the robots drive around and perceive objects (tasks) in the environment. For the transportation of an object from one location to another, two robots have to collaborate. Therefore, two roles are associated with a task: a helper role and a collector role. The coordination among agents is then based on the exchange of tokens, which are associated with the roles of a task. The tokens can be exchanged between agents and whoever owns a token can participate in the task in the associated role. Lau et al. [LWL07] present an approach grounded on immunity-based coordination (see Subsection 3.3.5), using a fleet of AGVs for material handling in an automated warehouse, where the AGVs are considered as immune cells that handle and complete the tasks, considered as antigens.

Valckenaers et al. [VKvB+01, VHG+07] present an approach for manufacturing control systems grounded on pheromone-based coordination (see Subsection 3.3.6), which uses resource agents corresponding to physical parts, order agents representing tasks in the underlying system, and product agents holding the process and product knowledge to assure the correct making of a product. Weyns et al. [WBH06] present an approach grounded on field-based coordination (see Subsection 3.3.7). They describe an AGV system where loads have to be transported in a warehouse. In this approach, new transportation requests emit fields into the environment that attract idle AGVs. To avoid multiple AGVs driving towards the same pickup location, AGVs emit repulsive fields. The AGVs combine the received fields and follow the gradient of the combined fields, that guide them towards pickup locations of transportation requests. Due to the fields the AGVs also continuously reconsider the situation of the environment and task assignment is delayed until the load is picked, which improves the flexibility of the system. Based on the experiments made in [WBH08] the authors show that in their settings field-based coordination outperforms market-based coordination. Similar to pheromone-based coordination approaches and field-based coordination approaches, our decentralized solution method based on infochemical-based coordination (see later Section 7.2), makes use of the advantages of environment-mediated coordination using stigmergy.

Unfortunately, the advantages of all decentralized agent-based solution methods grounded on decentralized coordination models do not come without disadvantages. Apparently, the biggest disadvantage of these solution methods is the lack of an optimal solution, as one can never be sure how optimal a MAS solution is (see *Problem 2* at page 7). Furthermore, because agents make their own decisions autonomously at

runtime, a deep understanding of the problem or transparency of decisions made is not always present so that one cannot control the actions of a MAS, which is one reason that has hindered the adoption of MAS in industry. Because the agents have to coordinate their local activities and decisions, they are also not a good choice when communication is very expensive. Moreover, unexpected emergent (mis)behavior may occur that could cause further troubles.

7.1.3.3 Hybrid Solution Methods

Due to the advantages and disadvantages of both purely centralized as well as purely decentralized solution methods, a couple of hybrid solution methods have been proposed that combine and integrate the principles of both classical OR techniques and agent-based solution methods. However, the literature on these approaches shows that the combination and integration of centralized and decentralized solution methods may occur in different ways at different levels. Thus, we classify these hybrid solution methods according to their degree of decentralization, starting with distributed OR optimization approaches, which only distribute classical methods over a group of agents, to fully embedded optimization approaches, in which every agent runs its own optimization algorithm:

- **Distributed OR optimization:** The most centralized way of combining classical OR techniques and agent-based solution methods is to only distribute the classical centralized solution methods over a group of agents, by decomposing the overall optimization problem into subproblems that have to be solved by an agent. For instance, Hirayama [Hir06] solves the Generalized Mutual Assignment Problem, another formulation of the distributed task assignment problem, by a distributed solution protocol using Lagrangean decomposition and distributed constraint satisfaction, where the agents solve their individual optimization problems and their locally optimized solutions are coordinated through a distributed constraint satisfaction technique. In [HPD09] Holmgren et al. distribute the principles of the Dantzig-Wolfe decomposition, a classical optimization technique, for the solution of an integrated production, inventory, and distribution routing problem. They propose a coordinator agent that corresponds to the master problem and planner agents that represent the subproblems and assist the coordinator agent in its search for the global optimal solution. However, by distributing classical OR techniques, in the majority of cases there has to be made a tradeoff between the quality of a solution and the cost of finding the solution, while such approaches may even fail to find an optimal solution at all (cf. [Hir06]). Furthermore, these approaches are in general only suited for static PDPs again.

- **Centralized agent-based optimization:** A very similar way of combination, however with a more agent-oriented view, is to exploit the beneficial properties of the agent technology for a centralized optimization. An example is given by the approach proposed by Dorer and Calisti [DC05] for dynamic

transportation problems. This approach clusters the available vehicles according to their geographical regions and allocates one manager agent for the planning of all vehicles in a region. Based on an insertion heuristic, dynamically arriving requests are then in a first phase assigned to these manager agents by a centralized dispatcher agent. In a second phase, the manager agents use cyclic transfers [TP93] – a class of neighborhood search algorithms – to further optimize the initial, valid solution, before assigning the requests finally to the vehicles. Agents embedded in the vehicles, if any, are only used to provide sensory information to the dispatcher agent, such as information on the traffic situation or real-time tracking information, so that the latter can re-optimize the plan, if necessary. However, vehicle agents do not participate actively in the solution or optimization process.

Similarly, the approach proposed by Leong and Liu [LL06] in the first phase uses a centralized push forward insertion heuristic [Sol87] executed by a global planner agent to obtain an initial, valid solution. In a second phase, agents representing customers and vehicles are only used to jump out of local optima and to increase the probability of reaching a global optimum. However, the global planner agent is always in possession of global knowledge and is able to coordinate the operations of the other agents. Because this approach produces a static output, it is not able to cope with any dynamic requests or events. Approaches of this class moreover are not scalable w. r. t. an increasing number of requests and will fail if the centralized planner crashes.

- **A priori optimization with operational re-planning:** This class of approaches still uses classical OR techniques for a coarse planning, while agents are used for operational re-planning, i. e. for performing local adjustments of the initial plan in real-time to handle the actual conditions when and where the plan is executed. In most cases the a priori optimization is for a longer time period. For instance, Davidsson et al. [DPH07] rudimentary describe an approach for the solution to an inventory routing problem, in which a centralized agent solves a predicted problem every n-th time step by a CPLEX algorithm (which is based on a branch-and-bound algorithm). During operation, vehicle agents are however allowed to deviate from the initial plan, if necessary, to better react on dynamic events. However, the authors state that when the degree of predictability is too low, i. e. dynamic events occur not as predicted for the optimization, a purely decentralized approach is superior.

- **Hierarchical optimization:** In hierarchical optimization approaches, optimization of the solution takes place at different hierarchical levels. Usually, the aim of such approaches is to have a balanced mixture of the advantages of both centralized and decentralized solution methods. The objective is to construct a more flexible and faster solution compared to a fully centralized method but also an improvement of the solution constructed by fully decentralized methods. Fischer et al. [FKM94, FMPS95, Fis96] were one of the first

who have proposed such a hierarchical system by the MARS system. The system consists of two agent types on different levels, namely truck agents on the lower one and shipping company agents on the higher one. Shipping company agents are responsible for the allocation of incoming dynamic transportation requests to the truck agents by an extended CNP. As this allocation may become sub-optimal with the incoming of new requests, in times when no new requests arrive a truck agent runs a Simulated Trading heuristic [BHM96] with the other truck agents in order to improve the global solution. Shipping company agents on the higher level are additionally equipped with some global knowledge in order to further optimize the overall solution by cooperating horizontally with other shipping agents applying a market-based coordination model again. The TeleTruck approach presented in [BFV00] is an extension of the MARS system, incorporating heterogeneous agent types as well as more realistic constraints. Perugini et al. [PLSP03] improve the extended CNP by Fischer et al., to allow truck agents to place multiple possibly-conflicting bids for partial routes, which also helps to overcome the eager bidder problem [SKF02] to some extent. Leveled commitments [SL02] and decommitments [HP04] improve this solution further.

Mes et al. [Mvv07] present an approach consisting of four different types of agents. On the one hand vehicle agents on the lower level and fleet manager agents on the higher level, on the other side job agents (representing requests) on the lower level and shipper agents on the higher level. Vehicles are assigned to jobs based on market-based coordination using Vickrey auctions. The fleet manager agents collect and analyze auction and processing time data of all its vehicles and distribute the results to their associated vehicles when needed, which thus have access to more information than their own history only. The same applies to the shipper agents for all the requests issued by the shipper. Thus, the role of the higher level agent is rather to centralize information essential for the agents on the lower level agents, which may improve the coordination between agents. However, based on their global knowledge, shipper agents can also reallocate the transport capacity that has been acquired by the job agents or switch the execution order. Similarly, fleet manager agents can reassign vehicles to jobs to improve the profit of the fleet. Vehicle agents are also allowed to trade requests, if they have not been started already.

- **Embedded Optimization:** In contrast to all other approaches, the optimization can even be embedded only in the vehicle agents, without any centralized or a priori optimization. Kohout and Erol [KE99] present an approach, in which the vehicle agents compute their overcosts for inserting a dynamically arriving transportation request based on an adopted version of Solomon's insertion heuristic. The vehicle agents send their costs as quotes to the customer agents demanding a service, which in turn selects the vehicle with the lowest bid, implementing the CNP. If the vehicle can still serve this request at the quoted price, the customer is inserted to the vehicles route and the contract is

accepted. Otherwise, the customer agent tries to make such a contract with the next cheapest vehicle agent. In contrast to [FMPS95], there is no further real post-optimization in form of cooperation between the vehicle agents. However, as a kind of post-optimization, customer agents are allowed to stochastically request removal from a vehicle schedule and to search for a better contract. In its centralized version, this technique is generally known as 'swapping' [CDSS02], which yields significant improvements in this distributed implementation.

A similar but less sophisticated approach was taken by Bertelle et al. [BNOT09], even though their vehicle agents only use a standard insertion heuristic for computing the overcosts. The vehicle agent then, by contrast, broadcasts the overcosts to the other vehicles and determines based on a kind of leader election mechanism the vehicle with the minimal overcosts as winner. However, they make no performance comparisons to existing approaches, but the approach is expected to run poorly.

Again, Davidsson et al. [DPH07] rudimentary describe an approach for the solution to their inventory routing problem, in which every vehicle agent now executes a CPLEX algorithm for optimization for n time periods itself, before making decision for the current time period. Their experiments proved that the solution quality improved compared to a purely decentralized approach, however again, only if the prediction of future events is not too bad. A similar approach is presented in [BJ09]. Dynamic transportation requests are assigned to the vehicle agents based on the CNP, but the vehicle agent subsequently not only aim to optimize the global solution by exchanging requests, but also by optimizing their own routing costs by a 2-opt algorithm. If static requests are known a priori, the approach allows for a centralized assignment to the vehicle agents by a sweep heuristic [GM74]. Equally, [BJ09] use CNP for initial task allocation as well, whereas the vehicles use the Clarke-Wright savings algorithm [CW64] for the optimization of their local routes and an inter-opt local optimization method for exchanging requests with other vehicles to optimize the overall solution.

Apparently, the main advantage of such solutions is in its fine granularity and high scalability, while its main disadvantage stems from a considerable overhead in computation time and resource usage. The overhead in computation time is mostly due to more expensive agent communications when compared to a fully centralized solution, while the overhead in resource usage depends on the memory and processing footprint of an agent.

All of the hybrid approaches mentioned above prove experimentally (some also theoretically) that the combination of centralized and decentralized solution methods into hybrid solution methods outperforms purely centralized respectively decentralized solution methods in their considered application domains and for their assumed constraints. Thus, hybrid solution methods are estimated to achieve the best results with regard to dynamic PDPs (cf. [MSdWZ10]).

7.2 Instantiating the IBC Approach for the Solution to PDPs

By instantiating the IBC approach (see Chapter 4) in this section as well as the EIA approach (see Chapter 6) in the next section, we are able to realize a hybrid solution method to PDPs, which in its entirety can be classified as an hierarchical optimization approach. Whereas the decentralized coordination mechanism grounded on IBC is inspired by a biological paradigm and hence realizes an efficient self-organizing emergent solution to dynamic PDPs, the instantiation of the EIA approach improves the solution efficiency for dynamic (and stochastic) PDPs with recurring tasks, while respecting the challenges and constraints present for self-organizing emergent systems (see *Challenges 4–6* at page 8). Thus, the entire solution on the one hand is able to take dynamically emerging events efficiently into account as well as on the other hand works efficiently in case that the distributions of the recurring tasks are *not* known a priori, but are only revealed as the routing plan is being executed.

A biological paradigm that self-organizingly solves similar natural transportation problems as defined by PDPs is the pollination of flowers by honey bees (see Subsection 7.2.1). Based on this paradigm, we instantiate the IBC approach to the so-called Pollination-inspired Coordination (PIC) approach (see Subsection 7.2.2), which represents a decentralized coordination mechanism applicable for the solution to PDPs. At the end of this section (see Subsection 7.2.3), we present related work with regard to this biological paradigm.

7.2.1 Biological Inspiration: Pollination by Honey Bees

While in economy a PDP refers to the transfer of goods from pickup stations to delivery stations, in botany, pollination (cf. [PYL96]) refers to the transfer of pollen (grains) – the male gametes – from the anther – the pollen releasing part of the stamen – to the stigma – the sticky, pollen receiving part of the pistil – of flowering plants (see Figure 7.2). Transferring pollen is, however, only possible during the blooming period of flowers, representing natural time windows. While in economy the vehicles housed in a depot are the essential elements for the transfer of goods, in botany, honey bees housed in a hive are the essential elements for the transfer of pollen. Honey bees thereby solve the biological analog of a PDP in a self-organizing emergent manner. Thus, in the following, we investigate the principles behind pollination by honey bees in more detail and identify the function of infochemicals in this coordination process, in order to draw inspiration from this biological paradigm for the design of a decentralized, IBC-based coordination mechanism applicable in a self-organizing emergent MAS solution to PDPs.

7.2.1.1 Pollination

For a better understanding of the biological background, the stamen is the male sexual organ of a flower, the pistil is the female one that additionally contains the

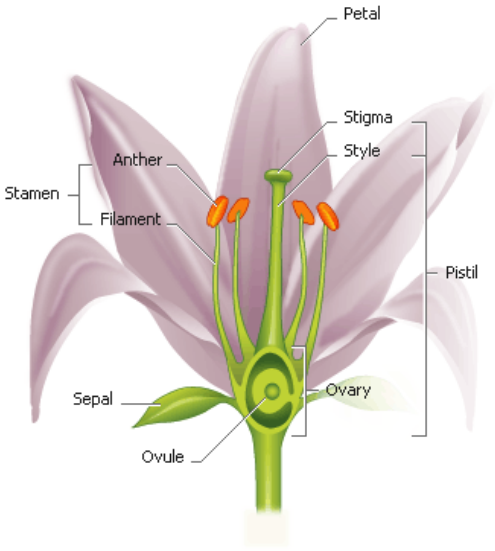


Figure 7.2: Main parts of a mature flower [Mic08]

ovary with an ovule, which in turn contains the female gametes (see Figure 7.2). The stigma is connected to the ovary by a style, through which a pollen tube may grow from a pollen grain to the ovary transporting the male gametes to the ovule, provided that the pollen grain lodges on the stigma and that pollen grain and flower are from the same species. When fertilized, the ovule eventually becomes a seed respectively a fruit.

A successful pollination is an important prerequisite for the reproduction of plants. In general, there exist two different types of pollination:

- **Self-pollenization/Self-pollination (*autogamy*):** The pollination of a (hermaphroditic) flower by its own pollen is called self-pollenization. If no external means are required for this act, the pollination process is called self-pollination. There exist certain plant species, for which this type of pollination is very frequent and volitional, whereas other plant species exhibit diverse mechanisms that make self-pollination improbable, e. g. different adolescences or chemical suppression substances (hormones). Autogamous plants are often located on islands or extreme sites such as desserts or the Artic, at which either sparse individuals exist or external means are naturally variable.

- **Cross-pollination (*allogamy*):** The pollination of a flower by the pollen of another flower either of the same plant (*geitonogamy*) or of another plant (*xenogamy*) is called cross-pollination. The intention of xenogamy is to increase the probability of new combinations of genotypes as well as to prevent inbreeding. Consequently, most plants are designed for cross-pollination, which is also why we draw inspiration from this type of pollination.

Cross-pollination usually requires a *pollinator*. A pollinator is an agent that transfers the pollen, whereas the plant that provides the pollen is called *pollenizer*. Over the years of convergent evolution, pollenizers have developed a couple of so-called pollination syndromes that resulted from the adaptation to their pollinators [FAW+04]. Thereby, one has to distinguish between biotic pollination syndromes, such as *zoophily*, and abiotic pollination syndromes, such as *anemophily* and *hydrophily* (cf. [FP79]):

- **Zoophily** refers to a biotic pollination syndrome by plants that is associated with the pollination by animals, which can be flying, crawling, or hopping insects, mammals, or even birds. Zoomophilous plant species often have some typical features or traits, such as stamina and pistils within the same flower, showy colors, sweet odors, or memorable structures (petals or sepals) that attract pollinators, as well as useful resources for pollinators, such as nectar, fats, oils, resins, or dosses, that serve as a kind of reward for their visit. During the collection of these resources by a pollinator, pollen grains from the flower's anther cling to the body of the pollinator, often due to sticky pollen appendages or a structured topology of the outer pollen layer. The pollinator then unconsciously transfers these pollen grains to the stigma of the same or another flower, which accomplishes pollination.

- **Anemophily** refers to an abiotic distribution of pollen by wind. Anemophilous plant species are for instance grass species, ragweed, conifers, or sweet chestnuts. These plants often have feathery stigmas in order to catch as much pollen as possible, whereas the pollen grains often are small and lightweight or equipped with airbags. Anemophilous plant species do not need showy features. Thus, their flowers often are unimpressive, numerous, small, and densely crowded.

- **Hydrophily** refers to the abiotic distribution of pollen by water. This type of pollination is quite rare and mainly used by water plants living in both fresh water and salt water. One distinguishes between pollen transfer under or on the water respectively above the water.

Pollination syndromes increase the specialization of a plant species regarding its pollination, which has two important benefits. On the one hand, the efficiency of pollination is increased. The reward provided to (living) pollinators may be very "expensive" to produce. Thus, plants take a proactive interest in maximal pollen transfer with minimal reward effort. Different pollinator species however display different pollen transfer efficiency, due to their size, shape, or behavior. Thus, specialized syndromes have a critical influence on the pollinator species selection and consequently on the overall pollination efficiency. On the other hand, specialized pollination syndromes increase the selective foraging behavior of pollinators, known as 'flower constancy'. To achieve an efficient pollen transfer, plants also require pollinators that only visit one flower type while bypassing other equally rewarding

flower types. Otherwise, pollen grains would lodge uselessly on stigmas of flowers of incompatible types. But the pollinators' primary aim is not pollination but the maximization of the success in their own reproduction by optimal foraging. The latter is measured by (1) energy gains per time unit, which requires the detection of essential nutrients (nectar, fats, oils), (2) minimal time for flower detection, and (3) short handling time during the flower visits. Surprisingly, many pollinator species show a high flower constancy. Although this flower constancy was first described by Aristotle, the reason why pollinators show flower constancy still remains unclear. The most accepted explanation for flower constancy invokes some sort of cognitive limitation on the pollinators' ability to effectively search for and/or remember multiple combinations of floral traits at the same time [GL05].

7.2.1.2 Honey Bees as Pollinators

About 80% of all plant pollinations are biotical [BN97]. From the remaining 20% abiotically pollinated plants, about 98% are pollinated by wind and only 2% percent by water. The reason for this unequal distribution for the benefit of biotical pollination is the huge amount of pollen needed for the pollination by wind or water, whereas there is only a small chance for a successful pollination. As this ratio is not quite efficient, most plants rely on biotic pollination, primarily by insects.

It is estimated that one third of the human food supply depends on insects, thereby mostly on bees [CWO91]. From all bee species, the honey bee is classified as the most ecological one. This bee species reaps this reputation first of all due to its production of honey and bee wax, but the major value of the honey bee is the pollination of agricultural crops such as fruit trees, vegetable, and forage plants, as well as many wild growing plants that prevent soil erosion, for instance. Honey bees have an extremely high flower constancy and pollen preference compared to other insects. In an experiment [Zan36], 91% of all tested pollen baskets contained only pollen from one flower type. Furthermore, honey bees either collect only pollen or only nectar during a flight.

In contrast to the vast majority of all bee species, honey bees are eusocial insects and live in colonies, such as ants. A single bee is not able to survive on its own, but only as a member of a collective. This collective is organized optimally and always consists of one breeding female queen, a large seasonally variable population of sterile female workers, and seasonally up to a few thousand male drones:

- **Queen:** The queen is the only breeding female in the colony and thus mother of all drones, workers, and future queens. The queen is lacking the 'tools' of the workers, such as a pollen basket, wax emitting glands, and a well developed honey craw. The food of a queen solely consists of a secretion named 'royal jelly' produced by the head grands of the workers.

- **Workers:** The female workers take over all tasks within a hive necessary for a smooth flow of life in the hive, but are not able to pair with each other and to breed. In contrast to ants, which do the same task their whole life, bees fill out

different functions in their short life (about 45 days). This contains cleaning the hive and honeycomb cells (day 1-4), feeding the larvae (day 5-10), stowing of pollen and nectar in the hive as well as ventilating the hive (day 11-13), building of honeycomb cells (day 14-17), guarding the hive (day 18-21), and finally from day 22 until their death the collection of food (particularly nectar and pollen) for the hive.

- **Drones:** The drones of the honey bee do not have a sting and are harmless; they also do not exhibit pollen baskets or wax glands and are not able to secrete royal jelly. In numbers, they are smaller than the female workers and it is their only job to pair with new queens. Immediately after pairing, which always occurs outside the hive while flying, the drones die.

Honey bees possess an extremely perfected communication system. On the one hand, all essential information for the organization of the hive is propagated by pheromones, secreted by the queen or the workers. These pheromones are used by the bees for the identification of locations such as food sources, swarming, recognition of the queen at its marriage flight, emission of warning signs, control of food stock, birth control in the colony by regulation of the lay activities of the queen, as well as temperature and humidity control within the hive (cf. [Fre87]). On the other hand, the bees are executing so-called bee dances. A bee dance is performed by the female workers when they return from food collection having discovered a fruitful food source (about 10% of the bees). By this dancing several types of information on the food sources are communicated. Firstly, the existence of an plentiful food source is announced, secondly, the odor of the food source is communicated, and thirdly, the location of the food source may be communicated. Two different types of dances are distinguished (cf. [See96]):

- **Round dance:** The round dance indicates a food source close to 100 meters. Thereby the walk of the bee describes a circle alternating clockwise and counter-clockwise. The more plentiful the food source, the more agile and longer is the dance. By the round dance no information on the direction is communicated. When the bees have learned that the food source is not far away and what odor it is of, they leave the hive and fly in wide circles until they find the food source. Honey bees determine their flight distance by orientating at passed environment elements.

- **Waggle dance:** The waggle dance is performed for food sources more far-off, sometimes up to 10 kilometers from the hive. Thereby the bee seems to walk a short way straight ahead and to return to its starting point by an arc. Actually, during the waggle phase the bee stands with its feet directly on the ground, as revealed by a slow-motion analysis. The return arc is performed alternating clockwise and counter-clockwise. The angle of the straight to the vertical corresponds to the angle to the sun the bees have to adhere to in order to arrive at the food source. The plentifulness of the food source is indicated by

the waggle fortitude the bee performs by waggling with its body. The distance to the food source finally is determined by the time span of the waggle phase.

Bees observing the waggle dance learn the distance, direction, and odor of these flowers and can convert the information into a flight to the specified flowers [CS91]. This complicated bee dance has been already described by Aristotle, but has not been exactly explained until von Frisch [vF67]. More recent research results (see [RGS+05]), however, extend the results of von Frisch and proof that the bee dance on its own is not enough to guide bees from the hive to the right food source. Indeed the bees are guided to the right food area by the waggle dance, but upon their arriving in this area they start to search for the odors of the flowers, to locate and approach their target exactly.

Because the odors of flowers benefit the flowers as well as the honey bees, from the chemoecological point of view, they are classified as synomones (cf. Subsection 4.1.1). These synomones enable honey bees to efficiently identify and approach appropriate flowers, whereby they unconsciously transfer pollen grains between flowers of the same species. Thus, these synomones ensure the fulfillment of this biological PDP. This biological paradigm furthermore demonstrates how the simple local behavior of the two species results in a global behavior – the reproduction of plants and bees – beneficial to both.

7.2.2 Pollination-inspired Coordination

The purpose of drawing inspiration from this biological paradigm is not to copy the exact behavior of both species plants and honey bees, but rather to adapt the simple but beneficial coordination principles grounded on IBC to realize self-organizing emergent MASs able to solve PDPs in economy. Thus, the pollination of flowers by honey bees represents beside foraging in ant colonies (see Subsection 4.5.1) the second fruitful inspiration for decentralized coordination by means of infochemicals that is used for the solution of computational problems. Even though honey bees use pheromones for intraspecific interactions in order to organize life in the hive, the essential infochemicals in the pollination process are the synomones emitted by flowers, mediating the interspecific interactions between the flowers and the honey bees and enabling the latter to efficiently identify and approach the former. However, some aspects of the pollination process become apparent, which in the nature usually carry no weight, but, transferred to the solution to PDPs in economy, have an essential impact on the solution efficiency or solution design:

- **Misdirection of pollinators by outdated synomones:** Synomones guide pollinators to appropriate flowers that provide and usually also need pollen grains from other flowers. However, not until a synomone is completely evaporated, i. e. its current concentration in the air has fallen below the threshold concentration of the specific pollinators (see Subsection 4.1.3.3), it will cease guiding pollinators to its emitting flower, even if the emitting flower is

already pollinated. Transferring this aspect to the solution to PDPs in economy, vehicles representing pollinators would be attracted to pickup or delivery stations representing pollenizers, although they do not provide or desire any more goods, which deteriorates the solution efficiency significantly.

- **Attraction of multiple pollinators to the same flower:** Often crowds of pollinators attracted by the same synomones converge on a single flower, although only one pollinator would suffice to pollinate the flower respectively has enough space to pollinate the flower. Thus, only the pollinator arriving first will be successful in pollinating the flower, while the rest has to go away empty-handed or has to remain waiting on other parts of the plant. Transferring this aspect to the solution to PDPs in economy, this kind of undesired swarm movement greatly reduces the solution efficiency, as superfluous routes or waiting vehicles may generate high costs.

- **Guidance of pollinators to the hive:** A further aspect, which does not concern the solution efficiency but rather the design of the solution, represents the guidance of pollinators to their hive. Whereas in botany honey bees exactly remember the position of their hive in the environment by remembering their flight distance and passed environment elements[4], the guidance of vehicles to their depot representing the hive would require sophisticates routing mechanisms and high computational resources at every vehicle as well as fixed environment structures. However, for a simple design of self-organizing emergent solutions a coherent coordination model is required (cf. *Challenge 1* on page 6).

In order to cope with the potential inefficiencies and design difficulties, in the computational world we extend the coordination between pollinators and pollenizers by three additional types of infochemicals using the design guidelines proposed in Section 4.4:

- **Foraging allomones:** In order to avoid a misdirection of the computational pollinators (vehicles) by outdated digital synomones of already pollinated pollenizers (pickup stations), the pollenizers have to keep further pollinators off from visiting. As depicted in Table 4.3 on page 114, this requires a kind of predatory antagonistic relationship between these two types of agents in such situations. The design guidelines thus recommend to use *foraging allomones*, which are emitted by the flowers and immediately propagated through the environment having a detaining effect on the pollinators. Thus, an approaching pollinator perceiving the outdated synomones of this flower will be kept away due to the additionally perceived allomones and look for other pollenizers to visit.

[4] Even this property can be detrimental, if the hive is moved for instance only by one meter, as then returning honey bees will not find the hive any more!

- **Territorial pheromones:** In order to avoid an attraction of multiple pollinators to the same flower, a pollinator approaching a flower has to keep further pollinators off from visiting the same flower. As depicted in Table 4.2 again, this requires a kind of negative, egoistic, nonsocial relationship between agents of this type in such situations. The design guidelines thus recommend to use *territorial pheromones*, which are emitted by the pollinators and immediately propagated through in a small area around the emitting pollinator, including the information which flower it intends to visit. Thus, pollinators following in a similar direction to the same flower perceiving these pheromones will instantly switch to another flower.

- **Aggregation kairomones:** In order to guide pollinators the way back to the hive (depot), the hive has to attract the pollinators. As depicted in Table 4.3 on page 114, this requires a kind of outwards directed commensal relationship between these two agent types in such situations. The design guidelines thus recommend to use *aggregation kairomones*, which are emitted by the hive and propagated through the environment having an attracting effect on the pollinators. Thus, pollinators that do not perceive any suitable synomones in the environment any more, may return to the hive by following the gradient of the kairomones.

The experiments will demonstrate that the coordination by means of four different types of infochemicals significantly improves the solution efficiency. Therefore, in Subsection 7.2.2.1 we describe the conceptual model of PIC with all the used types of infochemicals as an instance of the IBC approach. Subsection 7.2.2.2 subsequently specifies the objective coordination in PIC, i. e. the coordination between the three types of agents, whereas Subsection 7.2.2.3 specifies the subjective coordination in PIC, i. e. the coordination within each of the three agent types.

7.2.2.1 Conceptual Model

Based on the inspiration and the aforementioned extensions, Figure 7.3 depicts a conceptual model of the PIC mechanism as an instantiation of the DIC model (see Figure 4.2). A `Pollenizer Agent`, a `Pollinator Agent`, and a `Hive Agent` instantiate an *agent* of the DIC model and are situated on a `Location` in a common `Environment`. The agents belong to separate *type*s (`Pollenizer Type` and `Pollinator Type` respectively), linked in such a manner that pollinator agents can observe `Synomone`s as well as `Foraging Allomone`s emitted by the `Flower`s of pollenizer agents. Synomones and allomones guide a pollinator agent from its current location to suitable locations of a pollenizer agent. `Territorial Pheromone`s emitted by pollinator agents themselves additionally support this guidance. Furthermore, `Aggregation Kairomone`s emitted by hive agents support the guidance of the pollinator agents to the hive agents. The emission of infochemicals is always a `Location-based Action`.

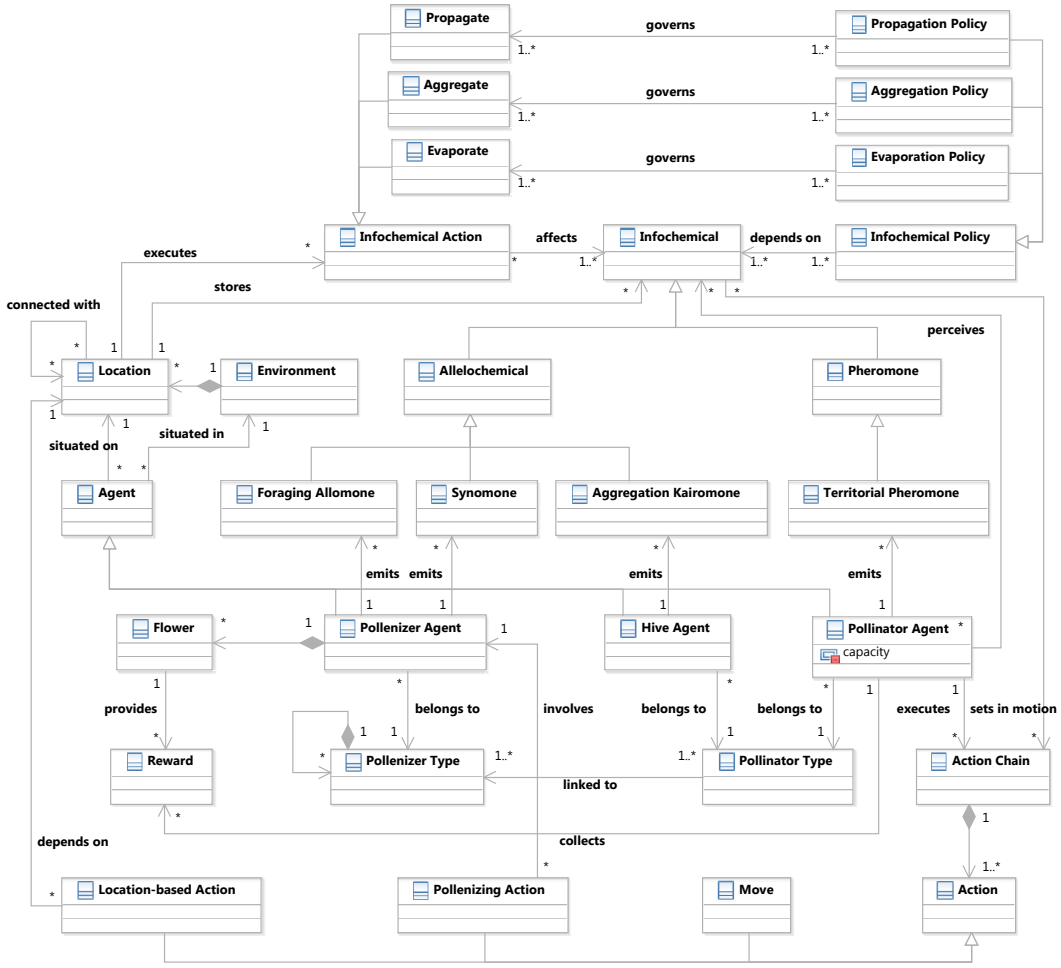


Figure 7.3: Conceptual model of PIC

Due to this bouquet of different `Infochemicals`, which are affected by several `Infochemical Actions` executed by the locations in the environment, pollinator agents now are able to `Move` efficiently through the environment according to the specific needs of pollenizer agents. The biological pollination shows us the need for having a chain of `Pollenizing Actions` in order to fulfill these needs, namely a pollinator agent needs to visit at least one more pollenizer agent after the initial visit to the first pollenizer agent. In our instantiation of the DIC model to PIC, we also use the concept of an `Action Chain` that mainly represents an intended sequence of interactions with pollenizer agents that all need to be performed to execute a task the system developer wants to achieve. An agent can be involved in several `Action Chains`, but we usually limit the number of "open" chains by a `capacity`. A flower provides a `Reward` (representing the estimated value of a reward for a pollinator in biology) for each pollenizing action performed.

7.2.2.2 Objective Coordination

In indirect coordination mechanisms such as PIC, the objective coordination, i. e. the coordination between the agents, is specified by defining the behavior of the coordination environment in response to a message, in this case the behavior of the infochemical environment in response to an infochemical. Therefore, we first have to specify the four different types of infochemicals in PIC more formally. The behavior of the infochemical environment is then defined by specifying the behavior of a location part of the environment.

Synomone

A synomone ς emitted by a flower fl of a pollenizer agent Ag_{pnz} is specified as

$$\varsigma = (\gamma_\varsigma, \gamma_\varsigma^{thresh}, \delta_\varsigma, \epsilon_\varsigma, fl, ppg, dpg, rc) \tag{7.13}$$

where

- $\gamma_\varsigma \in \Re$ is the current concentration of ς
- $\gamma_\varsigma^{thresh} \in \Re$ is the threshold concentration of ς
- $\delta_\varsigma \in \Re$ is the diffusion coefficient of ς
- ϵ_ς is the emitting agent Ag_{pnz} the flower fl belongs to
- ppg is the number of pollen grains provided by fl
- dpg is the number of pollen grains desired by fl
- rc is the concentration of the reward provided by fl

In the case of a synomone in PIC, fl, ppg, dpg, and rc together make up the individual information ψ encapsulated by an infochemical (see Definition 4.1).

Foraging Allomone

A foraging allomone α emitted by a flower fl of a pollenizer agent Ag_{pnz} is specified as

$$\alpha = (\gamma_\alpha, \gamma_\alpha^{thresh}, \delta_\alpha, \epsilon_\alpha, fl) \tag{7.14}$$

where

- $\gamma_\alpha \in \Re$ is the current concentration of α
- $\gamma_\alpha^{thresh} \in \Re$ is the threshold concentration of α
- $\delta_\alpha \in \Re$ is the diffusion coefficient of α
- ϵ_α is the emitting agent Ag_{pnz} the flower fl belongs to

In the case of a foraging allomone in PIC, only fl makes up the individual information ψ encapsulated by an infochemical (see Definition 4.1).

Territorial Pheromone

A territorial pheromone φ emitted by a pollinator agent Ag_{pto} is specified as

$$\varphi = (\gamma_\varphi, \gamma_\varphi^{thresh}, \delta_\varphi, \epsilon_\varphi, \chi) \tag{7.15}$$

where

- $\gamma_\varphi \in \Re$ is the current concentration of φ
- $\gamma_\varphi^{thresh} \in \Re$ is the threshold concentration of φ
- $\delta_\varphi \in \Re$ is the diffusion coefficient of φ
- ϵ_φ is the emitting agent Ag_{pto}
- χ is the infochemical Ag_{pto} is following at the time of emission

In the case of a territorial pheromone in PIC, χ (which can be both a synomone or a kairomone) makes up the individual information ψ encapsulated by an infochemical (see Definition 4.1).

Aggregation Kairomone

An aggregation kairomone κ emitted by a hive agent Ag_{hv} is specified as

$$\kappa = (\gamma_\kappa, \gamma_\kappa^{thresh}, \delta_\kappa, \epsilon_\kappa) \tag{7.16}$$

where

- $\gamma_\kappa \in \Re$ is the current concentration of κ
- $\gamma_\kappa^{thresh} \in \Re$ is the threshold concentration of κ
- $\delta_\kappa \in \Re$ is the diffusion coefficient of κ
- ϵ_κ is the emitting agent Ag_{hv}

In the case of an aggregation kairomone in PIC, no additional individual information is required.

Behavior of a Location

Every location part of the infochemical environment is able to execute three different actions affecting an infochemical. In more detail, a location can propagate, aggregate, and evaporate an infochemical, depending on specific infochemical policies. To specify these policies, we will instantiate the propagation, aggregation, and evaporation function defined by DIC.

In general, these actions are combined as follows: If an agent situated at a location l emits an infochemical ι as an instance of any of the aforementioned infochemical types, l will first clone ι by the number of neighboring locations connected by an inbound connection and then start to propagate these clones to each such neighboring location (see Equation 7.17). Upon the reception of a clone, a neighboring location aggregates the clone with its already stored infochemicals (see Equation 7.18). If the aggregation process is successful, i. e. the clone could be stored, this location will similarly start to clone and propagate the clone to all of its neighboring locations connected by inbound connections and so on. This will produce a kind of gradient field spanned around the emitter of the infochemical. Independently of the propagation and aggregation of infochemicals, at every time step a location additionally evaporates every infochemical currently stored.

Propagation Function Equation 7.17 instantiates the propagation function defined in Definition 4.6 for the PIC mechanism. While an infochemical ι is propagated from a location l_1 to a location l_2, its concentration γ_ι is decreased depending on the distance $d_{l_1 l_2}$ between the two locations (measured in e. g. in meters, hops, etc.) and the diffusion coefficient δ_ι. If the concentration of ι at the location l_2 would be below the threshold concentration γ_ι^{thresh} defined for this type of infochemical, ι will not be stored at l_2. The propagation function in PIC does neither make use of an propagation factor nor a propagation rate.

$$prop : \iota_{l_1} \mapsto \begin{cases} \iota_{l_2} & \text{if } \gamma_{\iota_{l_2}} \geq \gamma_\iota^{thresh} \text{ where } \gamma_{\iota_{l_2}} = \gamma_{\iota_{l_1}} - d_{l_1 l_2} \cdot \delta_\iota \\ \emptyset & \text{if } \gamma_{\iota_{l_2}} < \gamma_\iota^{thresh} \text{ where } \gamma_{\iota_{l_2}} = \gamma_{\iota_{l_1}} - d_{l_1 l_2} \cdot \delta_\iota \end{cases} \tag{7.17}$$

Aggregation Function Equation 7.18 instantiates the aggregation function defined in Definition 4.7. In PIC, two infochemicals ι_1 and ι_2 of the same emitting agent ϵ present at location l will be aggregated to ι_1, if the concentration of ι_1 is higher than the concentration of ι_2. In case that both infochemicals are allomones, additionally the emitting flower fl has to be the same. Otherwise, ι_1 will be discarded and ι_2 remains at the location l. Infochemicals of different emitters as well as different infochemical types of the same emitter, such as a synomone and an allomone, can be stored in parallel at one location. In other words, there is always at most one instance of a certain infochemical type of a certain emitter stored at a certain location. Thus, a pollinator agent following a certain synomone will always be guided to the synomone's emitter on the shortest path, without the comparison of identical synomones.

$$aggr : \iota_{1,l}, \iota_{2,l} \mapsto \begin{cases} \iota_{1,l} & \text{if } (\iota_{1,l}, \iota_{1,2} \in \{\mathcal{P}, \mathcal{K}, \mathcal{S}\} \wedge \gamma_{\iota_{1,l}} > \gamma_{\iota_{2,l}} \wedge \epsilon_{\iota_1} = \epsilon_{\iota_2}) \\ & \vee (\iota_{1,l}, \iota_{1,2} \in \mathcal{A} \wedge \gamma_{\iota_{1,l}} > \gamma_{\iota_{2,l}} \wedge \epsilon_{\iota_1} = \epsilon_{\iota_2} \wedge fl_{\iota_1} = fl_{\iota_2}) \\ \iota_{2,l} & \text{if } (\iota_{1,l}, \iota_{1,2} \in \{\mathcal{P}, \mathcal{K}, \mathcal{S}\} \wedge \gamma_{\iota_{1,l}} \leq \gamma_{\iota_{2,l}} \wedge \epsilon_{\iota_1} = \epsilon_{\iota_2}) \\ & \vee (\iota_{1,l}, \iota_{1,2} \in \mathcal{A} \wedge \gamma_{\iota_{1,l}} \leq \gamma_{\iota_{2,l}} \wedge \epsilon_{\iota_1} = \epsilon_{\iota_2} \wedge fl_{\iota_1} = fl_{\iota_2}) \end{cases} \tag{7.18}$$

Evaporation Function Equation 7.19 instantiates the evaporation function defined in Definition 4.8. In PIC, an infochemical ι stored at location l decreases its concentration γ_ι between iteration t and $t+1$ depending its diffusion coefficient δ_ι and on the evaporation factor ef defined for the type of ι. If the concentration of ι at iteration $t+1$ would be below the threshold concentration γ_ι^{thresh} defined for this type of infochemical, ι will be removed from l.

$$evap : \iota_{t,l} \mapsto \begin{cases} \iota_{t+1,l} & \text{if } \gamma_{\iota_{t+1,l}} \geq \gamma_\iota^{thresh} \text{ where } \gamma_{\iota_{t+1,l}} = \gamma_{\iota_{t,l}} \cdot \delta_\iota \cdot ef_\iota \\ \emptyset & \text{if } \gamma_{\iota_{t+1,l}} < \gamma_\iota^{thresh} \text{ where } \gamma_{\iota_{t+1,l}} = \gamma_{\iota_{t,l}} \cdot \delta_\iota \cdot ef_\iota \end{cases} \tag{7.19}$$

The concrete values for the evaporation factor ef, the diffusion coefficient δ, and the threshold concentration γ^{thresh} of an infochemical have a significant influence on the entire solution. As higher ef for a given ι, as longer the lifetime of ι. Similarly, as higher δ_ι, as longer the lifetime of ι, but as smaller its propagation range. As higher γ_ι^{thresh}, as shorter the lifetime of ι as well as smaller the propagation range. Please note, PIC does neither make use of an evaporation rate, i. e. in PIC an evaporation occurs in every iteration, nor a propagation factor, i. e. in PIC not only a fraction of an infochemical is propagation but always the entire infochemical, nor a propagation rate, i. e. in PIC the propagation of an infochemicals takes places immediately.

7.2.2.3 Subjective Coordination

The subjective coordination, i. e. the coordination within an agent, is specified by defining the local behavior of each agent type part of the PIC mechanism.

Specification of Hive Agents

Algorithm 7.1 specifies the simple local behavior of a hive agent Ag_{hv}, which the latter executes in every iteration t. Ag_{hv} will emit a kairomone κ, if the time since the last emission of a kairomone, measured by a counter t_{last}, has reached the specified emission rate emr_{hv} of hive agents. If a kairomone is emitted, t_{last} will be reset to zero. In any case, t_{last} is incremented in every iteration.

Algorithm 7.1 Local behavior of a hive agent

Input: t
Output: $\emptyset$
if $t_{last} \geq emr_{hv}$ **then**
2: emit(κ)
$t_{last} \leftarrow 0$
4: $t_{last} \leftarrow t_{last} + 1$

Specification of Pollenizer Agents

Algorithm 7.2 specifies the simple local behavior of a pollenizer agent Ag_{pnz}, which the latter executes in every iteration t. Basically, Ag_{pnz} checks for every flower fl of its set of associated flowers Fl, if fl has been already pollinated, i. e. the number of pollen grains desired (dpg) by fl and the number of pollen grains provided (ppg) by fl is zero, so that it can be removed from the set of associated flowers (see lines 2–3). Otherwise, and if fl is currently in its blooming period representing the time windows of a task, fl is encouraged to bloom further on (see lines 4–5).

Algorithm 7.2 Local behavior of a pollenizer agent

Input: t
Output: $\emptyset$
for all $fl \in Fl$ **do**
2: **if** $|dpg_{fl}| = 0$ **and** $|ppg_{fl}| = 0$ **then**
$Fl \leftarrow Fl \backslash \{fl\}$
4: **else if** $t \geq t_{fl}^{start}$ **and** $t \leq t_{fl}^{end}$ **then**
$bloom(fl)$

Algorithm 7.3 specifies the behavior of a flower fl during its blooming period. It first checks, if a visitor, i. e. a pollinator agent Ag_{pto}, is present at its own location. In case that an appropriate visitor is present (see lines 2–5), an allomone α will be emitted, if it has not been emitted so far and if the number of pollen grains requested by the pollinator agent (rpg) are at least as much as the number of pollen grains provided by the flower (ppg), or the number of pollen grains offered by the pollinator agent (opg) are at least as much as the number of pollen grains desired by the flower (dpg). In other words, the flower will emit an allomone in order to deter other pollinator agents, as soon as a visiting pollinator agent completely fulfills the flower's needs. In any way, the flower will receive respectively provide the pollen grains pg offered respectively requested by the pollinator agent.

In case that no appropriate pollinator agent is currently present (see lines 7–10), a synomone ς will be emitted, if the flower still desires or provides pollen grains and if the time since the last emission of a synomone t_{last} has reached the specified emission rate emr_{pnz} of pollenizer agents. If a synomone has been emitted, t_{last} is reset to zero. In any case, t_{last} is incremented in every iteration.

Algorithm 7.3 Local behavior of a flower (BLOOM)

Input: $\emptyset$
Output: $\emptyset$
if $l_{fl} \cap l_{Ag_{pto}} \neq \emptyset$ **then**
if $\neg allomonesEmitted$ **and** $(|rpg_{Ag_{pto}}| \geq |ppg_{fl}|$ **or** $|opg_{Ag_{pto}}| \geq |dpg_{fl})|$ **then**
emit(α)
allomonesEmitted $\leftarrow$ **true**
5: *receive*(*opg*) **or** *provide*(*rpg*)
else
if $(|dpg_{fl})| > 0$ **or** $|ppg_{fl}| > 0)$ **and** $t_{last} \geq emr_{pnz}$ **then**
emit(ς)
$t_{last} \leftarrow 0$
10: $t_{last} \leftarrow t_{last} + 1$

Specification of Pollinator Agents

Algorithm 7.4 specifies the local behavior of a pollinator agent Ag_{pto}, which the latter executes in every iteration t. If Ag_{pto} is currently not situated on a location but is moving between two locations on a connection, it will proceed moving to the next location (see line 2). If Ag_{pto} is situated on a location and is currently pollinating a flower, it will consequently proceed with the pollination (see line 5). Otherwise, i. e. if Ag_{pto} is situated on a location and does not pollinate a flower, and if even no pollenizer agent Ag_{pnz} is situated on this location, Ag_{pto} will start to inspect l for appropriate infochemicals to follow (see line 8), i. e. it will perceive infochemicals possibly stored at this location that guide it the way to the next pollenizer or hive agent (see later Algorithm 7.5).

However, in case that an Ag_{pnz} is situated on the same location, Ag_{pto} checks for every flower fl of Ag_{pnz}, if it is currently pollinated by another pollinator agent or if it can pollinate the flower itself (see lines 10–11). If Ag_{pto} was following the infochemical (synomone) χ and fl is the emitting flower of this infochemical (see lines 12–14), Ag_{pto} has reached its target and will stop checking the flowers but start to pollinate its targeted flower fl_{target} (see line 19). If Ag_{pto} was currently not following any infochemical χ, however, it will also pollinate fl, if this flower matches the pollination criteria by chance (see lines 15–17), i. e. the pollenizer super type of fl is linked to its pollinator type[5], it has not visited fl yet, and either Ag_{pto} has not

[5] A pollenizer super type represents a genus as a main taxonomic rank in nature, whereas a pollenizer type represents a species. Due to evolution and the pollination syndroms (see Subsection 7.2.1), a pollinator of a certain species (pollinator type) is not able to pollinate flowers of every genus.

started an action chain Ac of the pollenizer type of fl yet, while fl provides pollen grains, or Ag_{pto} has started an action chain of the pollenizer type of fl already, while fl desires pollen grains. If none of the flowers of Ag_{pnz} fulfills these criteria, Ag_{pto} will start to inspect l for appropriate infochemicals to follow as well (see line 21).

Algorithm 7.4 Local behavior of a pollinator agent

Input: t
Output: $\emptyset$
if $l_{Ag_{pto}} = \emptyset$ **then**
  do proceed moving
else
  if pollinating **then**
5: **do** proceed pollinating
  else
    if $\forall Ag_{pnz} \in A : l_{Ag_{pnz}} \cap l_{Ag_{pto}} = \emptyset$ **then**
      do inspect($l_{Ag_{pto}}$)
    else
10: **for all** $fl \in Fl_{Ag_{pnz}}$ **do**
        if $\forall Ag_{pto} \in A : l_{Ag_{pto}} \cap l_{fl} = \emptyset$ **then**
          if $\chi \neq \emptyset$ **and** $fl = fl_{\chi}$ **then**
            $fl_{target} \leftarrow fl$
            break for
15: **else if** $\chi = \emptyset$ **and** $\theta_{fl}^{super} \cap \{\theta_{Ag_{pto}}^{linked}\} \neq \emptyset$ **and** $fl \notin Fl_{Ag_{pto}}^{visited}$ **and** ($|Ac_{Ag_{pto}}^{\theta_{fl}}| = 0$ **and** $|ppg_{fl}| > 0$ **or** $|Ac_{Ag_{pto}}^{\theta_{fl}}| = 1$ **and** $|dpg_{fl}| > 0$) **then**
            $fl_{target} \leftarrow fl$
            break for
      if $fl_{target} \neq \emptyset$ **then**
        pollinate(fl_{target})
20: **else**
        do inspect($l_{Ag_{pto}}$)

The inspection of a location l along with the local decision making, which infochemical to follow, is very critical for an efficient coordination. Ag_{pto} has to decide very quickly, based only on the locally observable information, which infochemical to follow in order to act efficiently for itself but also for the entire system. This is exacerbated by the fact that Ag_{pto} is allowed to handle multiple action chains Ac in parallel up to its capacity, i. e. $|Ac_{Ag_{pto}}| \leq cap$, but only one action chain of a certain pollenizer type θ in parallel, i. e. $|Ac_{Ag_{pto}}^{\theta}| = \{0, 1\}$. Note, the pollenizer type of any action chain is determined by the type of the flower the first pollenizing action was executed with.

Thus, we base the local decision mechanism of a pollinator agent Ag_{pto} on the estimated utility of following an infochemical to its emitter. Basically, the calculation of this utility in general is governed by the following policies, similar as in nature:

- The nearer an agent is to the location of the emitter of an infochemical, the higher is the utility of this infochemical. Thus, nearer emitters are privileged compared to emitters more far off, which promotes flexibility.
- The longer the time an agent follows an infochemical, the higher is the utility

to follow this infochemical further. This is in contrast a tribute to the effort that resulted from following the gradient of this infochemical up to the current location, which promotes stability.

In case of a synomone ς emitted by a flower fl of type θ, the utility $u(\varsigma)$ additionally depends on a couple of further policies:

- If Ag_{pto} has not started an action chain $Ac^{\theta_{fl}}_{Ag_{pto}}$, yet:
  - If Ag_{pto} perceives a foraging allomone α emitted by fl, which indicates that the information included in ς is already outdated, the moving to fl will have no more utility for Ag_{pto} and ς can be excluded from the decision making.
  - If Ag_{pto} perceives a territorial pheromone φ of a pollinator agent Ag'_{pto} with $Ag_{pto} \neq Ag'_{pto}$, which indicates that Ag'_{pto} is already approaching fl, the moving to fl will have similarly no more utility for Ag_{pto} and ς can be excluded from the decision making as well.
- If Ag_{pto} has already started an action chain $Ac^{\theta_{fl}}_{Ag_{pto}}$:
  - Following ς has to be given priority. This favors the processing or even closing of action chains in contrast to the starting of new action chains.

These policies are incorporated into different utility functions a pollinator agent will use for its decision making. Definition 7.3 defines the utility of following a given synomone ς. According to the aforementioned policies, the utility u of following ς is calculated based on the sum of the current concentration γ_ς and the follow time ft_ς, i. e. the time Ag_{pto} was already following ς up to the current location, adjusted by a given factor η. However, the utility will only be positive, if Ag_{pto} has not opened an action chain of the type θ_{fl}, the number of opened action chains is smaller than the capacity cap of Ag_{pto}, and ς includes the individual information that fl provides pollen grains. Otherwise, the utility of following ς will be zero. The utility function furthermore assumes that inappropriate synomones are already excluded from decision making and do not have to be evaluated.

Definition 7.3 (Utility of synomones)
Let ς be a synomone, fl its emitting flower, θ_{fl} the pollenizer type of fl, and ft_ς the follow time. The utility u for a pollinator agent Ag_{pto} of following ς is defined as

$$u(\varsigma) := \begin{cases} \gamma_\varsigma + \eta \cdot ft_\varsigma, & \text{if } |Ac^{\theta_{fl}}_{Ag_{pto}}| = 0 \wedge |Ac_{Ag_{pto}}| < cap \wedge |ppg_{fl}| > 0 \\ 0, & \text{else} \end{cases}$$

In case that Ag_{pto} has already opened an action chain of the type θ_{fl} and ς includes the information that fl desires pollen grains, following ς is given priority.

However, the calculation of the utility of prioritized synomones remains the same (see Definition 7.4). In other words, this utility function assumes that if a prioritized synomone is present, it will be followed, regardless of other present synomones.

Definition 7.4 (Utility of prioritized synomones)
Let ς be a synomone, fl its emitting flower, θ_{fl} the pollenizer type of fl, and ft_{ς} the follow time. The utility u for a pollinator agent Ag_{pto} of following ς is defined as

$$u(\varsigma) := \begin{cases} \gamma_{\varsigma} + \eta \cdot ft_{\varsigma}, & \text{if } |Ac^{\theta_{fl}}_{Ag_{pto}}| = 1 \wedge |dpg_{fl}| > 0 \\ 0, & \text{else} \end{cases}$$

Finally, Definition 7.5 defines the utility of kairomones, which is calculated very simply.

Definition 7.5 (Utility of kairomones)
Let κ be a kairomone and ft_{κ} the follow time. The utility u for a pollinator agent Ag_{pto} of following κ is defined as

$$u(\kappa) := \gamma_{\kappa} + \eta \cdot ft_{\kappa}$$

Based on these policies and definitions, Algorithm 7.5 specifies the entire inspection process, by which Ag_{pto} identifies, which infochemical, i. e. which synomone or kairomone, to follow from its current location l. First, Ag_{pto} perceives from every outbound connection c_{out} of l, more specifically, the infochemical buffer of such a connection, all available infochemicals and stores them internally grouped by their type (see line 6), i. e. pheromones $\mathcal{P}_{c_{out}}$, allomones $\mathcal{A}_{c_{out}}$, kairomones $\mathcal{K}_{c_{out}}$, and synomones $\mathcal{S}_{c_{out}}$. Subsequently, Ag_{pto} checks for each synomone $\varsigma \in \mathcal{S}_{c_{out}}$, if the pollenizer super type of ς, more specifically of the emitting flower of ς, is linked to its own pollinator type so that it would be able in general to pollinate the emitter of ς (see lines 9–10). Otherwise, ς will be not considered for evaluation in this iteration. Then, Ag_{pto} checks if no pheromone $\varphi \in \mathcal{P}_{c_{out}}$ exists, which indicates that another pollinator agent is already approaching the emitting flower fl of ς (see lines 12–13), and if no allomone $\alpha \in \mathcal{A}_{c_{out}}$ exists, which indicates that ς is already outdated (see lines 14–15). In both cases, ς will not be considered for evaluation in this iteration as well. Furthermore, if Ag_{pto} has already opened an action chain $Ac^{\theta_{fl_{\varsigma}}}_{Ag_{pto}}$ of fl's pollenizer type, has not visited fl yet, and fl desires pollen grains, it will add ς to the set of prioritized synomones $\mathcal{S}_{c_{out},prio}$ (see lines 16–17). Ag_{pto} also checks for all perceived kairomones $\kappa \in \mathcal{K}_{c_{out}}$, if its pollinator type is linked to the type of κ, more specifically the emitting hive agent of κ (see lines 18–20). Otherwise, κ will be not considered for evaluation in this iteration.

Subsequently, Ag_{pto} evaluates the validated sets of prioritized synomones, (regular) synomones, and kairomones based on the utility functions defined above, and adds the results of these evaluations to the sets of evaluated prioritized synomones $\mathcal{S}_{eval,prio}$, evaluated synomones $\mathcal{S}_{eval}$, and evaluated kairomones $\mathcal{K}_{eval}$, respectively

Algorithm 7.5 Inspection algorithm (INSPECT)

Input: l
Output: $\emptyset$

$\mathcal{S}_{eval,prio} \leftarrow \emptyset$
$\mathcal{S}_{eval} \leftarrow \emptyset$
$\mathcal{K}_{eval} \leftarrow \emptyset$
$\chi_{old} \leftarrow \chi$
for all $c_{out} \in l$ **do**
  do perceive $\mathcal{P}_{c_{out}}, \mathcal{A}_{c_{out}}, \mathcal{K}_{c_{out}}, \mathcal{S}_{c_{out}}$ from c_{out}
  $\mathcal{S}_{c_{out},prio} \leftarrow \emptyset$
  for all $\varsigma \in \mathcal{S}_{c_{out}}$ **do**
    if $\theta^{super}_{fl_\varsigma} \cap \theta^{linked}_{Ag_{pto}} = \emptyset$ **then**
      $\mathcal{S}_{c_{out}} \leftarrow \mathcal{S}_{c_{out}} \backslash \{\varsigma\}$
    else
      if $\exists \varphi : \varphi \in \mathcal{P}_{c_{out}}$ **and** $fl_\varsigma = fl_{\chi_\varphi}$ **then**
        $\mathcal{S}_{c_{out}} \leftarrow \mathcal{S}_{c_{out}} \backslash \{\varsigma\}$
      if $\exists \alpha : \alpha \in \mathcal{A}_{c_{out}}$ **and** $fl_\varsigma = fl_\alpha$ **then**
        $\mathcal{S}_{c_{out}} \leftarrow \mathcal{S}_{c_{out}} \backslash \{\varsigma\}$
      if $|Ac^{\theta_{fl_\varsigma}}_{Ag_{pto}}| = 1$ **and** $fl_\varsigma \notin Fl^{visited}_{Ag_{pto}}$ **and** $|dpg_{fl_\varsigma}| > 0$ **then**
        $\mathcal{S}_{c_{out},prio} \leftarrow \mathcal{S}_{c_{out},prio} \cup \{\varsigma\}$
  for all $\kappa \in \mathcal{K}_{c_{out}}$ **do**
    if $\theta_{\epsilon_\kappa} \cap \theta^{linked}_{Ag_{pto}} = \emptyset$ **then**
      $\mathcal{K}_{c_{out}} \leftarrow \mathcal{K}_{c_{out}} \backslash \{\kappa\}$
  $\mathcal{S}_{eval,prio} \leftarrow \mathcal{S}_{eval,prio} \cup$ evaluate$(\mathcal{S}_{c_{out},prio})$
  $\mathcal{S}_{eval} \leftarrow \mathcal{S}_{eval} \cup$ evaluate$(\mathcal{S}_{c_{out}})$
  $\mathcal{K}_{eval} \leftarrow \mathcal{K}_{eval} \cup$ evaluate$(\mathcal{K}_{c_{out}})$
if $\mathcal{S}_{eval,prio} \neq \emptyset$ **then**
  $\chi \leftarrow \varsigma$ with $\varsigma \in \mathcal{S}_{eval,prio}$ **and** $u(\varsigma)$ is maximal
  do followPrioritizedSynomone(χ)
else if $\mathcal{S}_{eval} \neq \emptyset$ **then**
  $\chi \leftarrow \varsigma$ with $\varsigma \in \mathcal{S}_{eval}$ **and** $u(\varsigma)$ is maximal
  do followSynomone(χ)
else
  $\chi \leftarrow \emptyset$
  $ft \leftarrow 0$
  $ut \leftarrow ut + 1$
if $ut > it$ **then**
  if $l \cap l_{Ag_{hv}} = \emptyset$ **then**
    if $\mathcal{K}_{eval} \neq \emptyset$ **then**
      $\chi \leftarrow \kappa$ with $\kappa \in \mathcal{K}_{eval}$ **and** $u(\kappa)$ is maximal
      do followKairomone(χ)
  else
    $\chi \leftarrow \emptyset$

(see lines 21–23). Thus, at the end of this step, these three sets include triples consisting of an infochemical, its utility, and the outbound connection it has been perceived on, ranked by the evaluated utility of the respective infochemical.

Ag_{pto} subsequently checks, if the set of evaluated prioritized synomones is not empty. If this set is not empty, Ag_{pto} will identify the infochemical to follow χ by selecting the synomone ς with $u(\varsigma)$ is maximal out of the set of evaluated prioritized synomones (see lines 24–26). If the set of evaluated prioritized synomones is empty but the set of evaluated (regular) synomones is not empty, Ag_{pto} similarly will identify the infochemical to follow χ by selecting the synomone ς with $u(\varsigma)$ is maximal out of the set of evaluated synomones (see lines 27–29). If both sets are empty, Ag_{pto} apparently will not follow any synomone, reset the follow time ft, and augment the counter for unsuccessful inspection time ut (see lines 31–33).

If the unsuccessful inspection time ut exceeds a given threshold, specified by the idle time it, i. e. Ag_{pto} has not perceived any appropriate synomones to follow for a certain amount of time, Ag_{pto} checks, if there are appropriate kairomones to follow back to its hive, i. e. if the set of evaluated kairomones is not empty. Apparently, Ag_{pto} only checks for kairomones, if it is not already at its hive. If the set of evaluated kairomones is not empty, Ag_{pto} will identify the infochemical to follow χ by selecting the kairomone κ with $u(\kappa)$ is maximal out of the set of evaluated kairomones (see lines 34–38). In all other cases, there will be no infochemical to follow (see line 40).

Algorithm 7.6 specifies the following of a prioritized synomone χ. In order to follow χ, Ag_{pto} first checks, if the end location l' of the connection c_{out} on that χ has been perceived is not blocked[6], i. e. if no other pollinator agent is currently situated on this location (see lines 1–3). If l' is not blocked, Ag_{pto} will augment the ft, presumed it was already following this synomone before, and moves to l' (see lines 4–8). If l' is blocked (see lines 9–10) and Ag_{pto} sticks already for a while, i. e. a counter measuring a possible deadlock time dt exceeds a given threshold dt_{thresh}, Ag_{pto} performs an evasion maneuver (see later Algorithm 7.9). Whereas honey bees in nature usually have plenty of space in order to avoid collisions in the air or congestions in front of a flower, in the computational world the pollinator agents are permanently confronted with these problematic or inefficient situations. This requires the capability of performing an appropriate evasion maneuver by the pollinator agents, in case that the next location they intend to move to is blocked, i. e. already occupied by another pollinator agent. If the moving to any location has been successful, Ag_{pto} will reset both the deadlock time dt and the unsuccessful inspection time ut (see lines 11–13). Otherwise, the deadlock time will be augmented (see line 15).

Algorithm 7.7 specifies the behavior of an agent Ag_{pto} when following of a synomone χ. Following a synomone is very similar to the following of a prioritizes synomone, except that Ag_{pto} emits a pheromone φ in case that it intends to move to a location l' (line 8). Thus, other pollinator agents in the vicinity are informed about the intention of Ag_{pto}. When Ag_{pto} follows a prioritized synomone, emitting a

[6]This functionality has to be provided by the infochemical environment

Algorithm 7.6 Follow prioritized synomone (FOLLOWPRIORITIZEDSYNOMONE)

Input: χ
Output: $\emptyset$
get connection c_{out} of χ
get end location l' from c_{out}
if $\forall Ag'_{pto} \in A : l_{Ag'_{pto}} \cap l' = \emptyset$ **then**
  if $\chi = \chi_{old}$ **then**
5: $ft \leftarrow ft + 1$
  else
    $ft \leftarrow 1$
  do move(l')
else if $\exists Ag'_{pto} \in A : l_{Ag'_{pto}} \cap l' \neq \emptyset$ **and** $dt \geq dt_{thresh}$ **then**
10: **do** evasion
if *moved* **then**
  $dt \leftarrow 0$
  $ut \leftarrow 0$
else
15: $dt \leftarrow dt + 1$

pheromone is not necessary. Because the following of a kairomone does not require the emission of pheromones as well, the process of following kairomones is the same as following prioritized synomones and thus not explained here.

Algorithm 7.7 Follow synomone (FOLLOWSYNOMONE)

Input: χ
Output: $\emptyset$
get connection c_{out} of χ
get end location l' from c_{out}
if $\forall Ag'_{pto} \in A : l_{Ag'_{pto}} \cap l' = \emptyset$ **then**
  if $\chi = \chi_{old}$ **then**
5: $ft \leftarrow ft + 1$
  else
    $ft \leftarrow 1$
  do emit φ
  do move(l')
10: **else if** $\exists Ag'_{pto} \in A : l_{Ag'_{pto}} \cap l' \neq \emptyset$ **and** $dt \geq dt_{thresh}$ **then**
  do evasion
if *moved* **then**
  $dt \leftarrow 0$
  $ut \leftarrow 0$
15: **else**
  $dt \leftarrow dt + 1$

Algorithm 7.8 specifies the moving of Ag_{pto} from a location l to a neighboring location l'. If l' is currently blocked, the agent will remain at its current location (see line 7). Otherwise, the agent locks l' such that no other agent can move to this location, unlocks l such that other agents now can move to its old location, and finally moves to l' (see lines 2–5). The mechanism reports about the successfulness of moving to l'. The functionality of locking and unlocking an location by an agent

has to be provided by the infochemical environment. This, however, allows the infochemical environment to provide information if a location is blocked by an agent.

Algorithm 7.8 Move algorithm (MOVE)

Input: l'
Output: moved
if $\forall Ag'_{pto} \in A : l_{Ag'_{pto}} \cap l' = \emptyset$ **then**
2: $lock(l')$
$unlock(l)$
4: $l \leftarrow l'$ // Represents the moving from l to l'
moved $\leftarrow$ **true**
6: **else**
moved $\leftarrow$ **false**

Algorithm 7.9 finally specifies an evasion maneuver. Because a pollinator agent always follows the (prioritized) synomone respectively kairomone with the highest utility, in case of a required evasion the evasion mechanism first removes this element from the respective list of evaluated infochemicals. Algorithm 7.9 exemplary describes this process for evaluated synomones (see line 1). As long as a move has not been successful, i. e. the agent was not able to move to another location, and the set of evaluated synomones is not empty, Ag_{pto} will identify the next infochemical to follow χ by selecting the synomone ς with $u(\varsigma)$ is maximal out of the set of evaluated synomones (see lines 2–3). Similarly to the behavior described in Algorithm 7.7, Ag_{pto} tries to follow this synomone (see lines 4–12). If this is again not successful, Ag_{pto} removes this synomone from the set of evaluated synomones as well (see line 14) and tries to use the next one, until this set is empty or a move has been successful. The last chance for an evasion maneuver finally is to use a random outbound connection, in order to escape a deadlock situation (see lines 15–18). If all neighboring locations are however blocked by other agents, there is apparently no chance to move and Ag_{pto} will remain at its current location at least until the next iteration.

Due to this general, problem-independent modeling, PIC is applicable for a wide field of problem classes that require the self-organizing coordination between multiple autonomous components of homogeneous and heterogeneous agent types. PIC enables robust and flexible solutions in the face of dynamic changes. The agents therefore have to be situated in a logical or physical environment, which may be extended with the needed infrastructure (for propagation, evaporation, etc.), whereas the environment structure may represent a part of or even the entire problem that has to be solved. Spatial movement of the components is supported, whereas information about their spatial locations is indirectly exchanged.

7.2.3 Related Work

The decentralized PIC mechanism presented in the last subsection is based on the DIC model but takes its biological inspiration originally from the pollination of flowers by honey bees. This emphasizes the DIC model as a versatile and coherent

Algorithm 7.9 Evasion mechanism (EVASION)

Input: l
Output: $\emptyset$
$\mathcal{S}_{eval} \leftarrow \mathcal{S}_{eval} \backslash \{\varsigma\}$ with $u(\varsigma)$ is maximal
while $l_t = l_{t+1}$ **and** $\mathcal{S}_{eval} \neq \emptyset$ **do**
$\chi \leftarrow \varsigma$ with $\varsigma \in \mathcal{S}_{eval}$ and $u(\varsigma)$ is maximal
get connection c_{out} of ς
get end location l' from c_{out}
if $\forall Ag'_{pto} \in A : l_{Ag'_{pto}} \cap l' = \emptyset$ **then**
if $\chi = \chi_{old}$ **then**
$ft \leftarrow ft + 1$
else
$ft \leftarrow 1$
do emit φ
do move(l')
else
$\mathcal{S}_{eval} \leftarrow \mathcal{S}_{eval} \backslash \{\varsigma\}$ with $u(\varsigma)$ is maximal
if $\neg moved$ **then**
get random c_{out}
get end location l' from c_{out}
do move(l')

model for efficient decentralized coordination (see *Challenge 1*) providing a high expressiveness (see *Challenge 2*). The DIC design pattern and the corresponding design guidelines furthermore allow for a fast adaptation of the design in order to engineer efficient solutions (see *Challenge 3*), by using four different types of infochemicals within the same coordination mechanism. This will be proven by the experiments made in Section 8.3.

The elaborated communicative and evaluative methods and procedures of honey bees, however, have already served as a source of inspiration for a couple of other computational solution methods apart from PDPs. In recent years, especially the natural foraging behavior of honey bees, in more detail the waggle dances (see Paragraph 7.2.1.2), have inspired the development of new metaheuristics for the solution of combinatorial and numeric optimization problems, for instance the *Virtual Bee Algorithm* (VBA) [Yan05], the *Bees Algorithm* (BA) [PGK+06], the *Honey Bee Colony Algorithm* (HBCA) [CSLG06], the *Artificial Bee Colony* (ABC) algorithm [KB07], or the *Bee Colony-inspired Algorithm* (BCiA) [HD09]. The essence and main inspiration of these algorithms is the communication and broadcasting ability of a bee to some neighborhood bees so that they can 'know' as well as follow a bee to the best source, locations, or routes to complete the optimization task. However, in contrast to the PIC mechanism, these metaheuristics all suffer from the same problems as the metaheuristics mentioned in Subsection 7.1.3, such as their strict centralization, their limitation to a bounded problem size, and their limited capability to cope with highly dynamic events.

Inspired in a similar way by the waggle dances but with focus on telecommunication networks, Wedde et al. [WFZ04] present a routing algorithm for this problem

domain, called *BeeHive*. In this algorithm, bee agents travel through network regions called foraging zones. On their way their information on the network state is delivered for updating the local routing tables. This algorithm was extended in [Far09]. Inspired by the waggle dances as well, Nakrani and Tovey [NT04] present an algorithm for dynamic server allocation in Internet hosting centers.

In parallel to our first publications of PIC (see [KB06a, KB06b, KB06c]), the pollination of flowers by honey bees has also been used as a biological paradigm for another solution methods, in more detail for a swarm clustering algorithm [KRLM06]. This algorithm, however, uses two other natural facts of pollination: Firstly, the growth of a plant species depends on the region it grows in such that in very appropriate regions an agglomeration of these species can be observed. Secondly, the natural selection process selects better fitted plants in a region to survive. In the adopted clustering algorithm, each artificial bee thus is a simple agent as well. The bees will pick up the pollen of flowers with lowest growth and transport the pollen to a source where it will grow better. Each pollen grows in proportion to its neighboring flowers and after some iterations the natural selection will select the flower with the best growth of one species to survive and will sear others.

Decentralized autonomous solutions to the VRP and the PDP respectively are, however, not only subject of research to MASs. Also in the field of multi-robot research, decentralized autonomous solutions to similar problems are investigated (see e. g. [AKOS09]). For instance, in multi-robot routing, the problem consists of routing collaborating robots over available paths between target locations for some purpose, e. g. search-and-rescue in areas hit by disasters, surveillance of a facility, placement of sensors, delivery of parts, or localized measurements. Similar to the VRP, the objective is also to find a route for each robot, so that each target location is visited exactly once by exactly one robot (no waste of resources), all target locations are eventually visited by some robot (mission completeness), and the entire routing mission is accomplished successfully in the best possible way (optimization of certain performance measures). The most widely used coordination model in this area is, however again, market-based coordination, in particular auctions (see e. g. [LMK+05]). Also the coordination under limited communication constraints is investigated (e. g. [MML09]).

7.3 Instantiating the EIA Approach for Improving Solutions to PDPs

Even though the decentralized PIC mechanism presented in the last section enables flexible, robust, and scalable solutions to various instances of the dynamic PDP, we know from Section 5.1 and Subsection 7.1.3 that decentralized solution methods in general are not able to produce a solution of an optimal quality to a dynamic problem on their own. The main reason is that due to the absence of global knowledge the system elements tend to make suboptimal local decisions that result in globally inefficient solutions. In this section, we therefore instantiate the generic advisor

model presented in Section 6.2 for the use with and the efficiency improvement of PIC-based solutions to PDPs.

In Subsection 7.3.1 we first define a measure of efficiency, i. e. the quality of the solution, which we aim to improve by an EIA. This provides the foundation for all further improvement efforts. In Subsection 7.3.2 we then identify and analyze situations, in which local decisions of single agents lead to inefficiencies in the global behavior of the system with regard to the defined quality measure. These situations are not restricted to PIC-based solutions to PDPs only, but are common to decentralized solutions based on environment-mediated coordination mechanisms. Most of these situations may even appear in arbitrary self-organizing emergent solutions as well. For these situations, we present and classify a number of exceptions rules in Subsection 7.3.3, which an EIA can employ to provide advice to specific agents in certain situations for the improvement of the global solution. Finally, in Subsection 7.3.4 we instantiate the generic actions of the EIA defined in Section 6.2 for PIC-based solutions to PDPs.

7.3.1 Solution Quality

For PDPs, very often the quality of a solution is based on a single measure, such as minimizing the total/average travel costs of the transportation agents, minimizing the completion time, minimizing waiting times, balancing the workload over all transportation agents, maximizing the throughput, or minimizing time window violations. However, in this thesis we decided to use a quality function that in particular aims to combine energy use related measures, like the traveled distance, with other measures of interest. Thus, the quality $qual$ of a solution sol to a PDP is defined with regard to the improvements by the EIA as

$$qual(sol) = \frac{\varpi}{\lambda \cdot qual_{dist}(sol) + \mu \cdot qual_{order}(sol) + \nu \cdot qual_{tw}(sol)} \tag{7.20}$$

where

- ϖ, λ, μ and ν are weight parameters
- $qual_{dist}(sol)$ is the (costs for the) total distance traveled by all Ag_i according to sol
- $qual_{order}(sol) = \sum_{i=2}^{|as(sol)|} order_i$ is the penalty accumulated by sol for fulfilling tasks in a different order than their announcement to the system. Therefore, the difference between the start times of all pairs of tasks ta_1 and ta_2 is summed up, if the start time of ta_1 is before ta_2 but in sol ta_2 is finished before ta_1, i. e.

$$order_i = \begin{cases} t^{start}_{ta_{i-1}} - t^{start}_{ta_i} & \text{if } t^{start}_{ta_{i-1}} < t^{start}_{ta_i} \wedge t^{served}_{ta_{i-1}} > t^{served}_{ta_i} \\ 0 & \text{otherwise} \end{cases}$$

- $qual_{tw}(sol) = \sum_{i=1}^{|as(sol)|} tw_i$ is the penalty for sol for all tasks that are not completed within the time window for the task. Therefore, the difference between the finishing time for a task t^{served} and t^{end} is summed up, if the finishing time is later than t^{end}, i. e.

$$tw_i = \begin{cases} t_{ta_i}^{served} - t_{ta_i}^{end} & \text{if } t_{ta}^{served} > t_{ta}^{end} \\ 0 & \text{otherwise} \end{cases}$$

- $as : Sol \to 2^T$ is an assignment function that returns the set of tasks that are already assigned in a given solution defined as

$$as(sol) = \{ta_1, ta_2, \ldots, ta_k\}$$

where $(ta_i, Ag_i, t_i) \in sol$ with $ta_i \in T^{rec}$, $Ag_i \in A$, and $t_i \in Time$.

Obviously, the three measures of interest are not aligning well with each other, so that many solutions have rather similar *qual*-values, which makes improvement by the EIA even harder, since it is less likely to have a big difference in quality between the emergent solution and the optimal solution. Please note that penalizing for tasks not performed in order also favors making decisions based on local information, favoring the self-organizing emergent system, again. Note also that for this quality function higher values represent better solutions.

7.3.2 Inefficiencies in Solutions Based on Environment-Mediated Coordination

With regard to the efficiency measure presented above, we have extracted and analyzed several types of situations, in which local decisions of single agents lead to inefficiencies in the global solutions to dynamic PDPs. These situations are not restricted to PIC-based solutions only, but are common to decentralized solutions based on environment-mediated coordination mechanisms in general, as for instance also the one presented in [WBH06], which is grounded on field-based coordination (see Subsection 7.1.3.2). Most of these situations may even appear in arbitrary self-organizing emergent solutions as well. Reasons for the suboptimal local decisions are listed in Section 5.1. The EIA in general is designed to be able to detect these situations and to improve the local decision making by the agents.

7.3.2.1 Requests Handled by Inappropriate Agents

Due to their greediness, vehicle agents (called pollinator agents in PIC, but vehicle agents in general) try to serve any request right upon the perception of the corresponding synomones respectively attracting field, assumed that no allomones or pheromones respectively repelling fields hinder the agents. Because the agents in particular cannot reject or defer a request, usually, the idle vehicle agent, which is

closest to a pickup request, will try to serve it, disregarding other requests in the vicinity and without regard for requests that might appear at the agent's initial location or its vicinity in the near future. This results in unnecessary long routes and thus in an increase of total travel costs, as other vehicle agents might be in a much better position to serve the request. It also results in a violation of time windows, as another vehicle agent has to approach from a location more far off to serve the likely future request.

7.3.2.2 Attraction of Multiple Vehicle Agents

Due to their greediness, in many cases, two or more vehicle agents may initially start to approach the same station agent (called pollenizer agent in PIC, but station agent in general) to serve a pickup request. Even though all vehicle agents leave a pheromone trail respectively emit a repelling field that keeps other agents from approaching to the same target, this trail respectively field is restricted to a limited area around the emitting agent and can therefore only be perceived by other vehicle agents that are close by. If the vehicle agents approach from different directions to the same station agent, the trails respectively fields have obviously only a late effect, which results in unnecessary movements and consequently increases total travel costs as well as may violate time windows of subsequently appearing requests.

7.3.2.3 Overcrowding of Pathways

Due to their greediness, in small environments or environments including bottlenecks with regard to certain pathways, the announcement of a huge amount of transportation requests within a short period of time may instigate a huge number of vehicle agents to leave the depot immediately. In particular if the amount of requests exceeds the amount of vehicle agents, all agents will start to follow a different synomone respectively attracting field, because the pheromones respectively repelling fields emitted by each agent indicate different targets. Due to the service times at pickup stations, which depend on the load sizes of the requests, as well as the waiting times of the following agents, congestions emerge and the vehicle agents stand in each others way, which results in the violation of many time windows.

7.3.2.4 Oscillation of Vehicle Agents

Due to their greediness, vehicle agents may repeatedly shuttle between the depot and pickup stations, if the transportation requests appear at these stations with an adverse interval. In other words, the period of time between the occurrence of the transportation requests is higher than the period of time the vehicle agents require to move from a delivery station back to the depot. This results in an increase of total travel costs and sometimes to a violation of time windows.

7.3.2.5 No Consideration of Stochastic/Static Requests

Due to their reactiveness, vehicle agents will only start to move to a station agent, if the latter has emitted appropriate synomones respectively an attracting field. However, in stochastic PDPs or PDPs with a mixture of static and dynamic requests, a certain amount of requests have to be taken into account that are already known a priori. In particular for tasks with tight time windows, this results in a violation of the latter, because the vehicle agents first have to spend the time for moving to a station agent demanding the service of a stochastic/static request, which they could have saved, if they would have started prior to the perception of the corresponding synomones respectively attracting field.

7.3.2.6 Undetected Requests

Due to their reactiveness, vehicle agents situated at the depot that do not perceive any appropriate synomones respectively attracting fields (e. g. due to a too short propagation range) will remain in the depot, even if appropriate synomones respectively fields could be found in the immediate vicinity of the depot. The same holds for vehicle agents situated out of the depot that do not perceive any appropriate synomones respectively attracting fields, which in such cases will (after a given period of time) return to the depot. Thus, requests at worst may remain undetected, or will be detected at some remote period, which increases travel costs and violates time windows.

7.3.2.7 Unserved Requests

Due to their reactiveness, vehicle agents following a synomone respectively attracting field to a station agent in order to serve a pickup request, may decide to follow another perceived synomone respectively attracting field, assumed that its utility is higher respectively computational-field value is smaller. Thus, due to this reconsideration of the environment, the original request may remain unserved, in particular if other vehicle agents have already stopped following the synomones respectively attracting field to the original request, as they have perceived the pheromones respectively repelling field of the switching agent. This not only results in an increase of total travel costs, but also in a violation of time windows.

7.3.3 Classification of Exception Rules

The identified types of insufficiencies emerge from the local behavior of vehicle agents in the first instance. Thus, exception rules have to be identified that are able to advice the local behavior of these agents in order to improve the efficiency of the global system. Therefore, the underlying coordination principles of the used coordination model/mechanism have to be taken into account to find the best way to formulate the exception rules. Each coordination mechanism has different parameters and different starting points that have to be considered (see [DH06]). Of course, in any

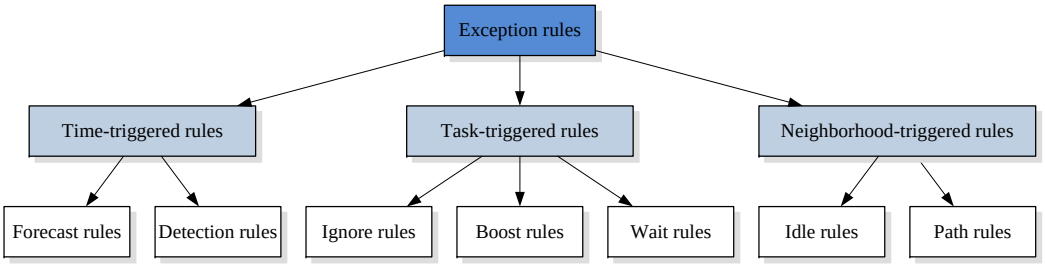


Figure 7.4: Exception rules to adapt the local behavior of agents in PIC

case, the agent model has to incorporate the application of exception rules during the evaluation of the agents' next actions. The exception rules defined below fit the DIC model tailored for the PIC mechanism and show exemplary how the formulation of rules can be approached. However, an application of these exception rules to other environment-mediated coordination mechanisms is conceivable, too.

In general, we distinguish three classes of exception rules derivable by an EIA in order to advice the local behavior of agents (see Figure 7.4): task-triggered rules, time-triggered rules, and neighborhood-triggered rules. As the name implies, *task-triggered rules* become activated in the rule base of an agent by the perception of a certain task, in the case of PIC by the perception of a certain synomone referring to a transportation request. In other words, a task must be present first in order to activate an exception rule of this class. By contrast, *time-triggered rules* do not require the presence of tasks but become activated due to the passing of a certain point in time or after a certain period of time. Apparently, these rules require the agents to have a common notion of time, e. g. the time measurement may start at the beginning of a day or the beginning of a run. Finally, *population-triggered rules* become activated neither by the presence of tasks nor the passing of a point in time, but only by the behavior of the other agents acting in the same environment, which is in the case of PIC recognized by the perception of pheromones. All three classes of exception rules are distinguished in a couple of more detailed rule types for PIC.

Task-triggered exception rules can be distinguished into the following three types:

- **Ignore rules:** This rule type forces an agent to ignore for a given period of time a perceived infochemical that is sufficiently similar to an abstracted infochemical. Thus, the utility of following this infochemical is set to zero. When the period of time has passed, the rule is no longer applied and the perceived infochemical is evaluated as it would normally have been. Ignore rules are very powerful and may be used by the EIA to deter an inappropriate or superfluous agent from serving a task (see 7.3.2.1 and 7.3.2.2, respectively), or to prevent the agent from oscillating (see 7.3.2.4). The structure of an ignore rule looks as follows:

```
If [perceived infochemical] is similar to [abstracted infochemical]
then ignore [perceived infochemical] for [period of time]
```

- **Boost rules:** This rule type forces an agent to boost for a given period of time and by a given percentage the utility of a perceived infochemical that is sufficiently similar to an abstracted infochemical. If the percentage is set to a value lower than 100%, the infochemical utility will be lowered accordingly. Consequently, ignore rules may be considered as a subclass of boost rules, in which the utility of an infochemical is boosted by 0%. When the period of time has passed, the rule is no longer applied and the perceived infochemical is evaluated as it would normally have been. Because boost rules increase the chance for an agent to follow a specific infochemical compared to other ones, they may be used by the EIA to guide an agent to a more optimal task in the vicinity (see 7.3.2.1) or to prevent an agent from changing its mind too often (see 7.3.2.7). The structure of a boost rule looks as follows:

  ```
  If [perceived infochemical] is similar to [abstracted infochemical]
  then boost [perceived infochemical] by [percentage] for [period of time]
  ```

- **Wait rules:** This rule type forces an agent to wait for the perception of an infochemical that is sufficiently similar to an abstracted infochemical. In other words, the agent ignores all infochemicals that are not sufficiently similar to the abstracted infochemical. However, the rule does not induce the agent to follow the infochemical it has waited for, but only to wait upon its perception. At this time the agent evaluates again the utility of every infochemical. In addition, a deadline has to be given, as otherwise the agent would wait until the end of the run instance and remain unproductive, if the task the abstracted infochemical refers to is not part of the run instance. Wait rules may be used by the EIA to deter an inappropriate agent from serving a couple of tasks (see 7.3.2.1) or to prevent an agent from leaving a location, e. g. the depot, too early (see 7.3.2.4). The structure of a wait rule looks as follows:

  ```
  If [perceived infochemical] is not similar to [abstracted infochemical]
  then ignore [perceived infochemical] until [point in time]
  ```

In contrast to the EIA, an agent has no notion of a task, as this is a high-level concept used only by the EIA. Instead, an abstracted infochemical has to be constructed by the EIA as an extraction from an identified task. This infochemical differs from a normal infochemical as it does not include complex objects such as the emitter of an infochemical, but rather simplified representations such as the location of the emitter. The similarity between a perceived infochemical and an abstracted infochemical can then be determined by an appropriate distance or similarity function, which may be similar to the one used in Section 6.2.3. Because in PIC the way the basic system reacts to its environment is based on the evaluation of infochemicals perceived by the agents, rules that influence the way the infochemicals are evaluated prove to be a very good starting point to influence DIC-based solutions in general.

Due to the fact that the basic movement capabilities of agents essentially require the presence of infochemicals stored at locations, in contrast to task-triggered rules, time-triggered rules additionally require the presence of a global environment map stored in each agents' data model, along with an appropriate routing algorithm that has to be incorporated by the agents. Otherwise, the agents will not be able to navigate to a given location or according to a given strategy. Time-triggered rules can be distinguished into two types:

- **Forecast rules:** This rule type forces an agent to move to an abstracted location or area, as soon as a given point in time is reached. However, the rule does not induce the agent to follow a specific route or to serve a specific task, which is still up to the agent's decision. As soon as the agent has arrived at the abstracted location, it has to wait for a given period of time. Thus, forecast rules may be used by the EIA to send an agent into an area, where the occurrence of a stochastic/static request is very likely at a specific time (see 7.3.2.5). Without the waiting constraint, the effect of the rule will get lost, if the dedicated request has not appeared yet, because the agent might then decide to serve another request instead. The structure of a forecast rule looks as follows:

```
If [current point in time] is equal to [point in time]
then move to [abstracted location] and wait for [period of time]
```

- **Detection rules:** This rule type forces an agent to move according to a given strategy, if the agent does not perceive any useful infochemical for a given period of time. A strategy may either include a couple of abstracted locations the agent has to visit or describe a certain movement pattern. Detection rules may be used by the EIA to let the agent search for yet undetected requests (see 7.3.2.6). The structure of a detection rule looks as follows:

```
If [perceived infochemical] is [null] for [period of time]
then move according to [strategy]
```

Finally, neighborhood-triggered rules can be distinguished into two types as well:

- **Idle rules:** This rule type forces an agent to remain idle for a given period of time, if the perceived amount of infochemicals, e. g. pheromones, exceeds a given threshold. Idle rules thus can be used by the EIA to prevent an overcrowding of pathways or to limit the number of agents currently acting in the environment, assuming that the agent is still in the depot (see 7.3.2.3). The structure of an idle rules looks as follows:

```
If [perceived amount of infochemicals] is higher than [threshold]
then idle for [period of time]
```

- **Path rules:** This rule type forces an agent to move according to a given strategy, which may again include a couple of abstracted locations the agent has to visit successively or describe a certain movement pattern, if the amount of perceived infochemicals exceeds a given threshold for a given period of time. Path rules may be used by the EIA in similar cases as idle rules (see 7.3.2.3), however, these rules are better suited for situations in which the agents are out of the depot. Thus, instead of idling and possibly blocking other agents, the agent may be advised to use an alternative path in order to avoid congestions. The structure of path rules looks as follows.

```
If [perceived amount of infochemicals]
   is higher than [threshold] for [period of time]
then move according to [strategy]
```

Although possible, a combination of different rule types has to be used carefully. For example, advising a vehicle agent to search for undetected requests by a detection rule may in turn result in an attraction of multiple vehicle agents to the same station agent (see inefficiency described in Subsection 7.3.2.2). Similarly, even though some rule types are more or less conflict free taken by itself, e. g. ignore rules, other rule types, in particular time-triggered rules, or a combination of different rule types may require some kind of deliberation. On the other hand, by a purposeful combination of different rule types the advice for an agent can be more specific. For example, by letting an agent ignore a certain infochemical and boost the utility of another one, the reaction of an agent may be anticipated very well. This emphasizes again the importance of simulating the set of derived rules by the EIA before it is send to the agents (see Section 6.3).

The instantiation of the EIA proposed in the next subsection will however only deal with one specific rule type, in more detail ignore rules, because handling requests by inappropriate agents has one of the highest potentials for optimization. Advising the system to let the best agent with regard to the global system optimality handle a request is already enough to reach a significant efficiency improvement for many scenarios. All other inefficiencies and exception rules are left for future optimization.

7.3.4 Instantiating the Generic Actions

The last step finally requires the instantiation of the generic actions *receive*, *transform*, *extract*, *optimize*, *derive*, and *send* of the EIA for PIC-based solutions to PDPs. Whereas this subsection provides the formal instantiation of the EIA actions, the technical realization of these actions is provided in Subsection 8.1.4.

7.3.4.1 Receive Local Agent Histories

As defined in Subsection 6.2.1, the main task of the action *receive* is to collect the local agent histories and to store them in the internal data structures of the advisor.

Two different types of data are principally of interest: environment data and agent data. Information about the environment in which the agents act has to be known in order to calculate the quality of solutions. Basically, this type of information can be reduced to the map of the environment as described in Subsection 7.1.2. Although we have assumed that each agent of the basic system must be able to collect data about its local behavior (see Section 6.2), i. e. its sensory input and its actions, an advisor not necessarily has to be interested in all basic actions an agent has performed during a run instance. For the purpose of advising the agents by ignore rules, the EIA is only interested in a few selected basic actions, which will naturally decrease the amount of data to be processed:

- **Choose an infochemical to follow:** The agents constantly re-assess the infochemicals they perceive and decide every time anew, which infochemical to follow. Switches in the infochemical to follow may lead to changes in the movement direction of the agent or even of other agents.
- **Service a station:** Whenever an agent reaches a station, it either picks up or delivers goods. Information about these actions together allows the identification of the individual tasks performed by each agent.
- **Move back to depot:** When an agent has been idle long enough, i. e. all perceived synomones during a certain period of time were not of interest or even no synomones have been perceived, it decides to move back to the depot.

As assumed in Section 6.2, there has to be a predefined point (place or time) at which the EIA must have access to the recorded agent data. Because in real world transportation scenarios the trucks or vehicles involved in a PDP return to the depot after all the work is done, in this instantiation this predefined point is at the end of each run. As soon as a run is completed, thus, the agent data is available in the EIAs situational knowledge and the action *receive* stores the data into the internal data structure $Dat_{EIA}^{Hist_A}$ of the EIA.

7.3.4.2 Transform Local Agent Histories into Global History

At the end of each run, the collected agent data is analyzed and transformed. As defined in Subsection 6.2.2, the transformation creates a global history of the form $(run_1, \ldots, run_k)$. This requires no more formal definition, as this action strongly depends on the realization of the basic system.

7.3.4.3 Extract Recurring Tasks from Global History

After a sufficient amount of data has been collected by the action *receive*, more specifically after a predefined number of run instances that serve as a learning time, and transformed by the action *transform*, recurring tasks are extracted by the action *extract*. Apparently, the instantiation of this action requires data of the current and from previous run instances. Having the run instances available in the form

$(run_1, \ldots, run_k)$, recurring task patterns can be identified using a similarity function for PDP tasks, sim^{PDP}, which instantiates the abstract similarity function defined in Definition 6.18. As already mentioned in Subsection 6.2.3, differences between recurring tasks can occur in each of the attributes of a task, which has an influence on the concrete specification of the similarity function as well as the centroid function later. Thus, in the case of PDPs, all the specific attributes of a transportation request (see Definition 7.1) have to be considered:

1. Localities of the request, i. e. s_{pickup} and $s_{delivery}$
2. The supply/demand q of $s_{pickup}/s_{delivery}$
3. Time windows, i. e. $t^{start}_{s_{pickup}}$ and $t^{start}_{s_{delivery}}$ as well as $t^{end}_{s_{pickup}}$ and $t^{end}_{s_{delivery}}$

Definition 7.6 (Similarity function for PDP tasks)
The similarity sim of two PDP tasks $ta_1, ta_2 \in T$ is determined by a function defined as

$$\begin{aligned} sim^{PDP}(ta_1, ta_2) \quad = \quad & \tau \cdot Ed(s_{pickup,ta_1}, s_{pickup,ta_2}) + \\ & \tau \cdot Ed(s_{delivery,ta_1}, s_{delivery,ta_2}) + \\ & \upsilon \cdot |q_{ta_1} - q_{ta_2}| + \\ & \omega \cdot (|t^{start}_{ta_1} - t^{start}_{ta_1}| + |t^{end}_{ta_1} - t^{end}_{ta_1}|) \end{aligned}$$

where

- τ, υ, and ω are constant weight factors
- $Ed : S \times S \rightarrow \mathbb{R}^+$ is the euclidean distance between the locations of two stations on an euclidean plane

The centroid of a task pattern respectively a cluster of PDP tasks is calculated by determining the center of the path between the locations of the euclidean plane, more specifically an average on each of the coordinates of the location, as well as the average of the loadsizes and time windows. The centroid function for PDP tasks instantiates the abstract centroid function in Definition 6.17.

Definition 7.7 (Centroid function for PDP tasks)
The centroid ce of a cluster of PDP tasks $T^{PDP} \subseteq T$ is determined by a function

$$\begin{aligned} cent^{PDP}(T^{PDP}) = \quad & (\frac{\sum_{ta \in T^{PDP}} s_{pickup,ta}}{|T^{PDP}|}, \frac{\sum_{ta \in T^{PDP}} s_{delivery,ta}}{|T^{PDP}|}, \\ & \frac{\sum_{ta \in T^{PDP}} q_{ta}}{|T^{PDP}|}, \frac{\sum_{ta \in T^{PDP}} t^{start}_{ta}}{|T^{PDP}|}, \frac{\sum_{ta \in T^{PDP}} t^{end}_{ta}}{|T^{PDP}|}) \end{aligned}$$

In general, there exist numerous classes of clustering algorithms (see e. g. [XW05]). An appropriate class of algorithms for the clustering of tasks, which consist of a

sequence of attributes (see Definition 6.1), in a sequence of run instances is *sequential clustering*. In this class of algorithms the tasks will be sequentially considered, i. e. one after the other, and inserted in one of the clusters. The advantage of sequential clustering compared to some other classes, e. g. hierarchical clustering, is that the clustering process can be performed even before the full data is present. Thus, the identification of task patterns can already start even if some data is still missing, which may speed up the clustering process.

Again, there exist numerous algorithms for the class of sequential clustering. Most of them require to be given initially the number of clusters it should produce, which is not appropriate for the clustering problem at hand. Additionally, many of them are not efficient enough in terms of memory and computation time [JMF99]. Sequential Leader Clustering (SLC) [Har75] constitutes an exception. It runs in linear time and does not require the designer to define how many clusters have to be built, as it dynamically adds new clusters, if necessary, and therefore does not limit the number of clusters a priori.

In more detail, in SLC the clustering of PDP tasks in a sequence of run instances $run_1, \ldots, run_k$ works as follow: If the clustering process has produced the clusters $Cl_1, \ldots, Cl_x$, with $ce_i \in Cl_i$ being the centroid of the cluster Cl_i, up to task ta in one of the run instances run_j, the similarity $sim^{PDP}(ta, ce_i)$ is calculated for all clusters $Cl_1, \ldots, Cl_x$, in order to put ta in one of the clusters $Cl_1, \ldots, Cl_x$. If cluster Cl_q has the highest similarity to ta, the task will be added to Cl_q, provided that $sim^{PDP}(ta, ce_q) \geq clustthresh$ for a given cluster threshold parameter $clustthresh$. The cluster threshold parameter controls the number and size of the clusters. A small threshold value will lead to a small number of large clusters, while a large threshold value will lead to a large number of small clusters. In the event of ta is added to Cl_q, it has to be checked, if the present centroid of Cl_q has to be changed. If ta is not similar enough to any of the clusters, i. e. $sim^{PDP}(ta, ce_i) < clustthresh \; \forall ce_i \in Cl_i \mid ce_i$ is centroid $\wedge \, Cl_i \in \{Cl_1, \ldots, Cl_x\}$, a new cluster $Cl_{x+1} = \{ta\}$ will be created and added to the list of clusters. A sequential leader clusterer is then a component performing SLC by a clustering function (see Definition 7.8).

Definition 7.8 (Sequential Leader Clusterer for PDP tasks)
The sequential leader clusterer for PDP tasks is defined as a tuple

$$SC = (LC, cent^{PDP}, sim^{PDP}, clustthresh, cf)$$

where

- LC is a list of clusters
- $cent^{PDP}$ is a centroid function for PDP tasks
- sim^{PDP} is a similarity function for PDP tasks
- $clustthresh$ is a constant threshold

- $cf : Cl \times T \rightarrow Cl$ is a clustering function that adds a task $ta \in T$ to a list of clusters Cl with

$$cf(Cl, ta) \rightarrow \begin{cases} (Cl_1, \ldots, Cl_q, \ldots, Cl_x) \text{ with } Cl_q = (Cl_q \cup ta, cent^{PDP}(Cl_q \cup ta)) \\ \quad \text{if } \forall p, 0 < p \leq x, p \neq q : sim^{PDP}(ta, ce_q) > sim^{PDP}(ta, ce_p) \\ \quad \wedge sim^{PDP}(ta, ce_q) > clustthresh \\ Cl \cup Cl_{x+1} \text{ with } Cl_{x+1} = (\{ta\}, ta) \\ \quad \text{otherwise} \end{cases}$$

The distillation of the medoids as most significant recurring tasks proceeds as described in Subsection 6.2.3.

7.3.4.4 Optimize Solution of Recurring Tasks

The instantiation of the action *optimize* requires to create a (nearly) optimal solution for the identified set of recurring tasks, which can then be used in comparison with the emergent solution of the basic MAS. An essential component for this job thus is a powerful optimizer that calculates optimal solutions for PDPs. Although the PDP is rather well researched, we were not able to find an efficient public-domain optimizer, which is why we had to implement a more simple branch-and-bound-based optimizer [LMSK63, Dak65] for the proof-of-concept. Because at the time of the *optimize* action all data about the tasks and the agents of the system is known due to the actions described before, the problem is no longer a dynamic PDP but has become a static PDP. Thus, a centralized branch-and-bound algorithm can be applied to compute exact solutions (see Subsection 7.1.3.1). Even though the proposed algorithm creates optimal solutions for the PDP, it is essentially not very sophisticated and as a result the size of PDPs it can tackle in an acceptable time is limited ($\leq$ 10 recurring tasks). However, the experiments in Subsection 8.4 will show that the optimizer was good enough to demonstrate the improvement abilities of the EIA approach.

Role of the Goal Function

An essential part of any branch-and-bound algorithm is the goal function to evaluate the goal value of a (partial) solution. If a good goal function is available that punishes bad solutions early on by assigning a bad value to them, the bounding can occur at an earlier stage and the calculation becomes much faster because fewer branches of the tree have to be explored. On the other hand, potentially good solutions must be rewarded with a higher quality value to make sure that they are not bounded at an early stage and thus get lost.

Obviously, the optimality of a solution always depends on the goal function that is used. Different goal functions will yield different optimal solutions. Therefore, it is possible to optimize a basic system for different parameters by exchanging the used goal function or by changing the weighting factors used in the function. This allows

for instance to punish solutions in which the vehicles travel a long distance, practically forcing all vehicles of the system to travel minimal routes. By changing the goal function it is also possible to create optimal solutions in which as few vehicles as possible leave the depot. Other examples are the optimization for low energy consumption, avoidance of routes that are used by other vehicles, emphasizing shorter distances over correct order, and so on. For the branch-and-bound-algorithm implemented for this thesis we have used the quality function defined in Equation 7.20 as goal function.

The Branch-and-Bound Optimization Algorithm

The branch-and-bound algorithm implemented for this thesis works as follows[7]: An initial, partial solution with no assigned tasks, i. e. $as(sol) = \emptyset$, is created as the root of the tree. As a next step, new solutions for each of the agents in A are branched from the root, which assigns one of the tasks of T^{rec} to one of the agents with the order 0. After that, the goal value of each of these partial solutions is calculated based on the goal function. For the best solution so far, the branching step is repeated and another task of T^{rec} is assigned to the agents with order 1. This process is repeated until a complete solution is found as a leaf of the tree, i. e. $as(sol) = T^{rec}$. A complete solution is required to compare the optimal and the actual solution. With it, also a lower bound for all other solutions is found, as we are only interested in solutions that are better than the one we already have. As a next step, the *bounding* is performed: Each partial solution, whose goal value is less than or equal to the goal value of the complete solution, is deleted. This way, solutions that are known to be worse than what was already discovered are not explored any further, which reduces the amount of branches, the memory consumption, and the computational complexity. As soon as no new branches can be created any more – because every branch has either been bounded or ends in a complete solution – the solution with the highest goal value is chosen as the optimal one.

To facilitate the calculation of optimal solutions, the developed branch-and-bound algorithm imposes certain conditions on the way the system works:

- The number of vehicles is known and constant.
- A distance of 1 can be covered by a vehicle in exactly 1 iteration.
- A pickup station starts to distribute synomones as soon as a task becomes valid, i. e. a task can be perceived only if its time window is already open.
- Each task can be handled by a single vehicle.
- Once a pickup has taken place, the goods are delivered immediately, i. e. a vehicle does not pickup goods of another task before delivering the ones it already loaded[8].

[7] For a more formal definition please refer to [LMSK63] and [Dak65]

[8] Technically, this means that there is no need to open more than one `ActionChain` at once.

All of these conditions are met in the current model of PIC and the EIA. If one of these constraints has to be alleviated, it is most likely that the goal/quality function defined at the beginning of this section will have to be changed.

As an And-Tree-Based search algorithm, the branch-and-bound algorithm deals with partial solutions for the problem instance, as described above. These partial solutions only include assignments for some of the tasks that are part of T^{rec}. The goal function ensures that a partial solution is never evaluated to a value worse than any of its derivatives including more assignments by simply ignoring the fact that some tasks have not been assigned to a vehicle yet. Values for $qual_{dist}$, $qual_{order}$, and $qual_{tw}$ are of course only calculated for tasks with assignments and the fraction ensures that the goal value of a partial solution is always an upper bound for any solution that contains the partial one. Therefore, the goal value of a child is always less than or equal to the parent.

Performance Considerations

The developed branch-and-bound algorithm is not very efficient and requires a lot of time and memory for larger problem instances (> 10 recurring tasks). The main factor that currently influences the performance of the algorithm is the number of nodes that have to be created in the search tree. This number depends on the number of tasks and the number of vehicles. Unfortunately, no better optimization algorithm was available and the development of such an algorithm is beyond the scope of this thesis. Nonetheless, future implementations of the EIA might use a better optimization algorithm, because there are some chances to reduce the number of nodes significantly.

As already mentioned above, a good goal function will enable the algorithm to apply bounds early. If potentially good partial solutions have a high goal value while poorer solutions are assigned a lower goal value, the bounding will prune a lot of low-value nodes as soon as the first solution is found resulting in less created nodes. Due to the definition of the algorithm the evaluation may never underestimate the potential of a partial solution.

A very easy way to improve the performance of the algorithm is to apply a simple heuristic that would allow for earlier bounding. The heuristic would be applied to partial solutions and establish a lower bound. Based on this lower bound, the pruning could be performed. An alternative for such a heuristic would be to add twice the minimal distance between two nodes in the environment for each unassigned task in the solution to the distance totally covered by the vehicles. Surely, no vehicle could reach the nodes where the requests for the task are located in less than twice this minimal distance.

Even though such optimizations allow for a better algorithm and thus enable the EIA to be used with larger problem instances as well, they were not incorporated as they are not within the research focus of the thesis.

7.3.4.5 Derive Rules from Optimal Solution

The instantiation of *derive* has to be able to detract an agent from a particular task respectively all similar tasks by ignore rules, if the optimal solution is sufficiently better than the emergent solution. To determine the *qual*-value of the emergent solution, we compute $qual_{dist}$ by using the direct distances between the emitter agents for the recurring tasks. The $qual_{order}$- and $qual_{tw}$-values are directly available out of the last run instance containing the particular recurring tasks, but as mentioned in Paragraph 6.2.5.1, $qual_{tw}$ can be heavily influenced by the non-recurring tasks of this run instance.

For the PIC-based system, we add the ignore rules to the synomone utility computation performed by an agent: any synomone, which is sufficiently similar to an abstracted synomone *ab* given in the condition of an exception rule, is not considered, respectively its utility is zero. *ab* consists of the elements $s_{pickup,ab}$, q_{ab} and t_{ab}^{start} of the task that should not be taken by the agent. To determine the similarity of a synomone representing a concrete task ta_1 to the abstracted synomone *ab*, Ag_{EIA} computes the value

$$\begin{aligned} dist_{syn}(ab, ta_1) \quad = \quad & Ed(s_{pickup,ab}, s_{pickup,ta_1}) + \\ & |q_{ab} - q_{ta_1}| + \\ & |t_{ab}^{start} - t_{ta_1}^{start}| \end{aligned}$$

If the distance to the abstracted synomone $dist_{syn}$ is below a given threshold $synthresh$, then the associated exception rule will be applied and the utility of the synomone representing ta_1 will be set to zero, effectively letting the agent ignore the perceived synomone. As mentioned in the last subsection, each exception rule is only applied for a limited time, which prevents tasks being ignored for too long, if the agent the EIA designated to serve the ignored task fails or is busy with servicing other tasks.

7.3.4.6 Send Rules to Agents

The derived ignore rules are stored in the internal data structure $dat_{EIA}^{Rule_A}$ that associates the rules with the respective agent. The next time a communication with the agent is possible, i. e. at the latest when the agent visits the depot again at the end of the next run, the rules are send to the agent.

7.4 Extension Aspects

Apart from the instantiation of the IBC approach by the PIC mechanism presented in Subsection 7.2.2 and the instantiation of the EIA approach presented in Subsection 7.3.4, a few aspects are conceivable, how to extend these instantiations in order to improve the overall efficiency of PIC-based self-organizing emergent MAS solutions to PDPs:

- **Incorporation of a re-optimization phase:** A central property of existing solution methods using hierarchical and embedded optimization (see Subsection 7.1.3.3) is the incorporation of a re-optimization phase. Thereby, an existing valid solution is further improved with regard to the global optimality, either by trading already assigned but not executed tasks between the vehicle agents, or by changing the execution sequence of assigned tasks within a single vehicle agent. The motivation for such a re-optimization results from the low flexibility of existing solution methods when dealing with tasks that can not immediately be started upon their assignment to vehicle agents – a drawback of direct communication, as e. g. employed by the CNP. In PIC-based solutions using indirect communication, tasks are only assigned to vehicle agents, if the vehicle agent has reached the pickup station and is able to immediately start the execution of the task. This preserves a high flexibility of the overall solution. Thus, a 're-optimization' of the solution would have to occur before this point in time. For instance a vehicle agent might evaluate the information included in pheromones indicating the presence of other agents approaching the same target, and come to the conclusion that this agent is better suited to fulfill the task, according to a quality measure that would have to be defined.

- **Variable reward concentration:** In its current specification, the PIC mechanism does not allow for the consideration of prioritized tasks. However, in real-world applications of PDPs, very often tasks with a higher urgency appear, for instance, food with a high perishableness, such as fresh fish or meet, has to be transported with a higher priority than packaged food. In order to introduce a priority concept, flowers can be equipped with the capability to variate the concentration (not the amount!) of the reward depending on the priority of the task. Synomones with a higher concentrated reward included could then be evaluated to a higher utility by a pollinator.

  A similar priority concept is even indirectly incorporated in the biological paradigm itself. The sugar concentration of pollen grains increases as longer the sun shines on them, by vaporizing the water molecules. Thus, pollen grains that have been neglected for a while by honey bees will be at the end more attractive than pollen grains that are collected immediately. Such a priority concept could also be used by the EIA, as it could advice a pollenizer agent to raise its offered reward concentration to become more attractive to pollinator agents, thus being served earlier.

- **Variable synomone emission concentration:** In its current specification, the PIC mechanism defines the initial concentration of an infochemical at the time of emission to be the same for all agents. However, flowers could be allowed to increase this emission concentration individually, if their provided or desired pollen grains are not picked up or delivered for a certain amount of time. Thus, the synomones would be propagated in a wider range having a higher utility, which possibly attracts more pollinator agents. Please note,

however, that due to such a variation the design of the global behavior of the system is much more complicated and unpredictable, while a starvation of the system might become possible.

- **Flexible synomone evaluation:** Even with the incorporation of a priority concept the utility for a pollinator agent to follow a specific synomone depends in the first instance on the information included in this synomone, along with the presence of specific allomones and pheromones. However, the utility of following a specific synomone could also be dependent on the presence of further synomones perceived at the current location. For instance, a pollinator then might not move in the direction of the synomone with the highest utility but in the direction with the highest aggregated utility of several synomones of different emitters, indicating an area with a lot of tasks to fulfill.

  Another option would be to take into account the delivery location of a task before moving to a pickup station, assumed that the corresponding synomones are already perceivable. Thus, more optimal local decisions could be made by pollinator agents, as they can calculate the optimality of following a synomone to a pickup station, giving them again a limited capability "to look into the future". A flexible synomone evaluation does not require deliberation capabilities by the agents but only to use more available information for their decision making, while remaining reactive.

- **Congestion detection:** Using more available information, more specifically information included in pheromones, can also be used for congestion detection. For instance, a pollinator agent could evaluate the amount of pheromones perceivable at its current location coming from a certain direction, in order to determine, if a congestion is likely to occur in this direction and another direction might be better in order to approach a target. To avoid congestions optimally, pollinators could be additionally equipped with an environment map. Congestion avoidance could also be advised by the EIA, for instance by path rules, which are triggered by the neighborhood (see Subsection 7.3.3).

- **Incorporation of further pollinator pheromones:** In the same way as pollenizer allomones inform pollinator agents that a task has been already assigned, even though synomones not yet evaporated are still perceivable, [-,+] pheromones emitted by a pollinator could inform other pollinators from a change in its mind with regard to its currently pursued target. Thus, the other pollinator agents could react more quickly to this change compared to the time the outdated pheromones with the old target require to evaporate. As the DIC model allows for the combination of different types of infochemicals, even of multiple instances of the same type, the additional pheromone could increase the overall flexibility of the system.

- **Realization of further exception rules:** Whereas in the last section only one type of task-triggered exception rules, more specifically ignore rules, has

been employed by the EIA, the realization and incorporation of further exception rule types and classes for the adaptation of the local behavior of agents will improve the performance of the global solution further. In particular with regard to problems incorporating tasks patterns with time windows, we expect time-triggered rules to achieve high efficiency improvements as well.

- **Consider pollenizer and pollinator types:** The notion of action chains is used to allow a pollinator agent to pickup goods from different pollenizer agents, more particularly from pollenizer agents of different species. The action chains separate these goods logically within the pollinator. A pollinator can therefore pickup several different goods on route (as long as its maximum capacity is not exceeded) and deliver them in an arbitrary order. So far, this ability is not used by the EIA, as it is unaware of different species and therefore assumes that a good is picked up and delivered right away. A more sophisticated implementation could make use of the concept and create advice that lets the pollenizers travel routes on which they mix pickup and delivery of different goods.

  A very similar concept the EIA is currently unaware of are pollinator types. A pollinator in nature is of a certain species and not all species of pollinators may be able to serve or visit all species of pollenizers. similar as in nature, some vehicle agents might not visit some of the station agents. The EIA would have to be augmented by letting the goal function sort out solutions, which assign agents of the wrong type to the tasks to accommodate this concept.

All of these options, however, are left for future optimization and have not been subject to the experimental evaluation presented in the next chapter.

7.5 Conclusion

In this chapter we have presented an instantiation of the IBC approach as well as the EIA approach in order to provide an efficient self-managing solution approach to dynamic (and stochastic) PDPs. Taken by itself, the instantiation of the IBC approach, i. e. the PIC mechanism, represents a decentralized solution method to PDPs. In contrast to existing decentralized solution methods, which are grounded either on market-based coordination (see [BWHM04, MvdHvH08, ZLL09]), token-based coordination(see [FINZ05]), immunity-based coordination (see [LWL07]), pheromone-based coordination (see [VKvB+01, VHG+07]), or field-based coordination (see [WBH06]), PIC is grounded on infochemical-based coordination and thus inherits all the beneficial advantages of IBC (cf. Table 4.4). In particular, the experiments regarding the IBC approach respectively the PIC mechanism made in Section 8.3 will demonstrate that the PIC mechanism allows for both a high solution efficiency as well as a high solution process efficiency. Due to the fact that there neither exist any commonly agreed benchmark data sets nor any commonly available simulation tools, we cannot compare these efficiencies to the efficiencies provided by existing

solutions. However, similar to all other decentralized solution methods, the runtime insufficiencies remain the same, i. e. the optimality of the solution cannot be guaranteed. For the improvement of the solution, hence, higher level approaches such as an EIA are required.

The instantiation of the EIA approach, taken by itself, does not represent a centralized solution method. It rather encapsulates a centralized solution method, in this instantiation a branch-and-bound algorithm. The centralized solution method is used to derive individual exception rules for the adaptation of the vehicle agents in order to improve the PIC-based self-organizing emergent MASs. Consequently, the EIA approach adopts the advantages and disadvantages of centralized solution methods. Although the EIA thus is able to calculate (nearly) optimal solutions, it usually takes a long time to compute this solution, while the complexity of the problem to solve is rather limited. This is one of the reasons, why the EIA only focuses on the set of recurring tasks for the optimization, which reduced the problem complexity.

Taken as a whole, the the IBC approach and the EIA approach together combine the principles of self-organization and self-adaptation. With regard to the PDP, it thus can be classified as a hybrid solution method, in more detail as a hierarchical optimization approach (see Subsection 7.1.3.3). Similar to all other hierarchical optimization approaches, the aim of our approach is to have a balanced mixture of the advantages of both centralized and decentralized solution methods, i. e. a more flexible and faster solution compared to the one constructed by a fully centralized method but also a more efficient solution compared to the one constructed by a fully decentralized method. In contrast to existing hierarchical optimization approaches (see [FMPS95, Mvv07]), our instantiation of the EIA approach is not in possession of global knowledge per definition, but learns the problem it has to optimize. Furthermore, it considers the low observability and poor controllability of self-organizing emergent MASs and is therefore able to adapt the local behavior of the vehicle agents.

Chapter 8

Experimental Evaluation

In order to prove that the models, mechanisms, and architectures developed in the previous chapters enable the design and operation of efficient self-organizing emergent systems, we have evaluated them experimentally in realistic case studies. Apparently, experiments with real self-organizing emergent systems for application domains such as PDPs would have been very time-consuming and costly, which again underlines the importance of a simulation tool in the design of these systems (cf. *Challenge 3*). Even though there exist plenty of simulation tools appropriate for the simulation of self-organizing emergent systems in general, simulation tools that in particular focus on the realization of environment-mediated, decentralized coordination models and mechanisms as well as their efficiency are (publicly) not available. This forced us to develop our own simulation tool, called **S**imulator for **E**fficie**N**t **S**elf-Organizing **E**mergent **S**ystems (SENSES).

Section 8.1 describes the software architecture of SENSES and explains the running of experiments as well as the realizations of the IBC and EIA approaches customized for PDPs in more detail. Subsequently, Section 8.2 describes two PDP case studies from the field of intralogistics, to which we have applied our concepts. Section 8.3 and Section 8.4 then describe the experiments we have executed regarding the two approaches, as well as present and analyze the experimental results. Overall, this will demonstrate the ability of our concepts to improve the efficiency of self-organizing emergent systems. Finally, Section 8.5 concludes this chapter.

8.1 Simulator for Efficient Self-Organizing Emergent Systems

Without doubts, there exist plenty of simulation tools, which are able to simulate self-organizing emergent systems. A non-exhaustive list of widely used simulation tools includes for instance Swarm [Swa08], [Rep08], SeSAm [SeS08], MASON [MAS08], Ascape [Asc08], StarLogo [Sta08], NetLogo [Net08], MadKit [Mad08], JADE [JAD08a], JADEX [Jad08b], Agentsheets [Age08], AnyLogic [Any08], breve [Bre08], CORMAS [COR08], ECHO [Hol02], and XRaptor [XRa08]. Whereas tools such as Agentsheets and AnyLogic are only commercially available, other tools are not platform independent, such a breve (Phyton/steve), CORMAS (SmallTalk), ECHO (C), or XRaptor (C++), and thus are excluded from the choice of tools.

Although the rest of these tools is mostly based on agent technology, they all have certain conceptual or architectural drawbacks, which is why we could not use or extend them for our purposes. Most of these simulation tools do not provide a clear separation between the problem/application model, the coordination model, and the solution model, respectively between objective and subjective coordination (see Section 3.1). However, this is essential for the development and testing of coordination models and in particular coordination mechanisms, as the latter have to be easily switched or adapted without changing other models. Also, most existing simulation tools rather only focus on the effectiveness of agent-based solutions but completely neglect the efficiency aspect of these solutions. These facts forced us to design and implement SENSES.

Please note that the purpose of SENSES in the first instance is to support the design and evaluation of efficient decentralized coordination models and mechanisms for different application domains. Thus, SENSES is designed to simulate the essential real-world aspects characterizing a given application domain. However, SENSES is not designed for a single application domain only simulating all of its aspects in full detail, even if it could be extended for that. The latter case would have forced us to use and extend a state-of-the-art commercial simulation tool for the focused application domain.

8.1.1 Software Architecture

SENSES is a Java-based simulation tool. Its two-tier software architecture comprises a presentation tier and an application tier. The latter is composed of a couple of components encapsulated in packages (see Figure 8.1), in order to provide a good extensibility for future simulation or evaluation demands. In particular, SENSES can be extended to simulate a variety of different emergent coordination models and mechanisms. This allows an engineer to switch between different problem models, coordination models, and corresponding coordination mechanisms without huge efforts. In more detail, the application tier of SENSES comprises the following components:

- **Core component:** This component comprises interfaces, abstract classes and information containers that are used by almost all other components. The core component does not use any other component and represents the core of SENSES.

- **Models component:** This component includes the Java realization of coordination models. Any coordination model used in SENSES, such as DIC, has to be included as a subcomponent in order to provide the basic functionality for the corresponding coordination mechanisms.

- **Mechanisms component:** Consequently, the mechanisms component is based on the models component. The mechanisms component includes the Java re-

alization of coordination mechanisms. Any coordination mechanism used in SENSES, such as PIC, has to be included as a subcomponent.

- **Problems component:** The problems component includes the Java realizations of problems that have to be solved by a self-organizing emergent solution. Any problem solved in SENSES, such as problems of the PDP domain, has to be included as a subcomponent. This component does not use any other components.

- **Solutions component:** The solutions component combines a coordination mechanism with a specific problem model. It manages the environment and the scenario that has to be simulated in an experiment. Beside the core component, the solutions component hence uses the models, mechanisms, and problems components.

- **Main component:** The main component is required to start and execute experiments in SENSES. The component uses the core, mechanisms, and solutions components and basically parses the configuration of an experiment, controls simulations, and gathers the experimental results.

- **EIA component:** The EIA component realizes the EIA to improve the basic solution to a problem. Apart from the problems component, the EIA component uses all other components, but in turn is not used by any of them. This shows even on the architectural level that the EIA is not required for a self-organizing emergent solution.

8.1.2 Running Experiments

SENSES is available as an executable jar-file. The syntax to run[1] SENSES is `java -jar SENSES.jar [options] [file]`. In order to run an experiment in SENSES, the experiment first has to be specified, which can be accomplished by two different ways:

1. By means of the configurator provided by the GUI of SENSES: This way is recommended for users with little experience in the coordination mechanism of interest and when the visual representation of the experiment is of importance. The GUI will be started, if the option to skip the GUI (`-n`) is not specified. By taking this way, several experiment parameters can be adapted as time goes on, while the effects of these adaptations can be understood visually at every time step. Beside this advantage one has to be aware of the fact that the visual representation naturally slows down the experiment execution.

2. By means of an experiment configuration `file`: This way is recommended for users that are more familiar with the coordination mechanism of interest as

[1] SENSES requires Java 6 or higher.

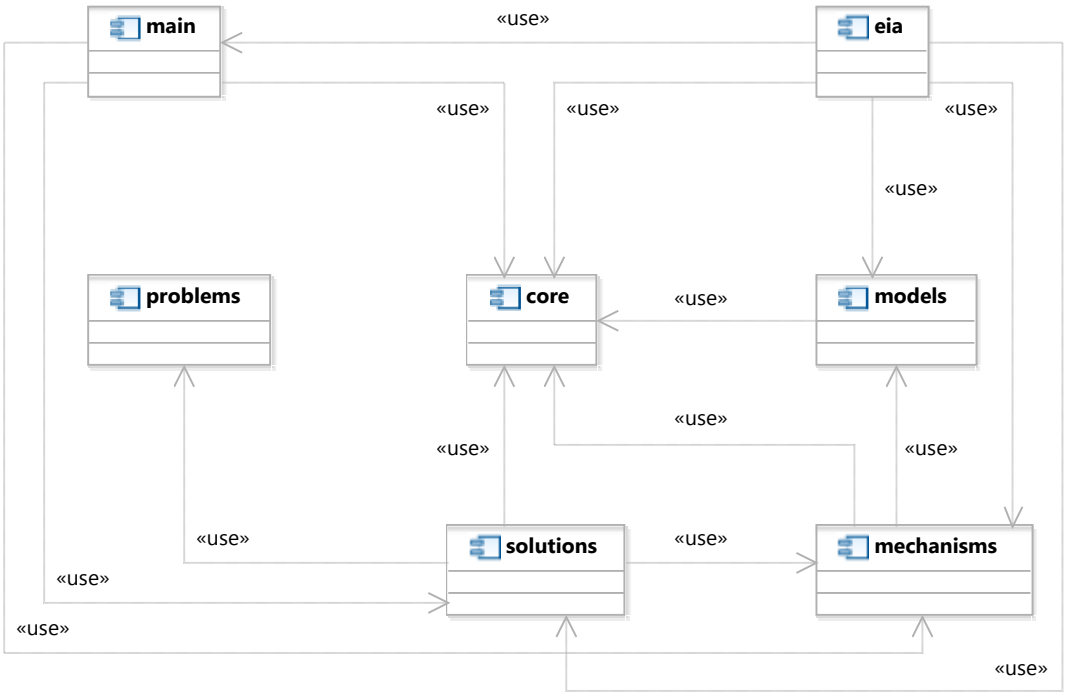


Figure 8.1: Components of SENSES

well as the effects of certain parameters and when the results of the experiment are what one is interested in. If the option `-n` is specified, the visual representation is skipped, which will speed up the experiment execution. The results of the experiment are aggregated by a result file. Nonetheless, the visual representation of the experiment execution can be used for this way as well, if required.

As a possible third way, an experiment configuration file may also be loaded afterwards via the GUI. However, this requires the specification of such a configuration file by one of the two ways described above. Thus, in the following, we focus on the experiment specification by an configuration file in XML (see Subsection 8.1.2.1), as this way is usually used for evaluation purposes. Consequently, environments (Subsection 8.1.2.2), scenarios (Subsection 8.1.2.3), and result files (Subsection 8.1.2.4) are specified in XML files as well. The structure of every XML file in turn is defined by corresponding XML Schema files.

8.1.2.1 Specification of an Experiment Configuration File

The XML Schema for an experiment configuration file is split into three parts on the highest level (see Listing A.1): one part deals with parameters for the experiment to run (*experiment specification*), one part deals with the logging behavior of SENSES during the experiment execution (*logging specification*), and one part deals

with default parameters (*default parameters specification*). One or more of these parts may be omitted in a configuration file, which allows for different variations of configuration files.

Experiment Specification

The specification of an experiment, i. e. the major part of a configuration file, first requires an `experiment name`, an `experiment type` (either `single-scenario`, i. e. the same scenario is simulated in one or more run(s), or `multiple-scenarios`, i. e. different scenarios are simulated in two or more run), and a specification

- if the solution has to be `optimized` by the means of an EIA
- if the experiment results have to be `recorded`, which will generate an XML file listing different measures at the end of a run
- if a `protocol` of the experiment has to be generated, which will store the experiment configuration as well as all the possibly randomly generated tasks that had to be fulfilled in the experiment, in order to be able to load and rerun the same experiment later again
- of the `number-of-runs` the experiment consists of
- of the `number-of-runs-learning`, which defines the number of runs the EIA learns before it starts to apply exception rules, if any.

Every `run` of an experiment is specified by a `run setup`, `run settings`, and `run parameters`. The `run setup` contains the following parameters:

- `problem-domain`: This parameter specifies the problem or application domain in which the self-organizing emergent MAS has to act.
- `environment`: The environment is a description of the world, more specifically of the map the agents are situated in and on which they can move. It hence contains information on the locations and the connections between the locations that make up the environment (see Paragraph 8.1.2.2 below for the specification of an environment).
- `scenario`: The scenario specifies the problem that has to be solved in more detail. It contains general information on the sequence of tasks that have to be fulfilled, such as the location where and the time when a task becomes available, the type of the task, the location where and the time when it has to be fulfilled at the latest, its probability to occur in the run, and domain specific properties, such as e. g. the capacity required to fulfill the task. Furthermore, the scenario includes general information on the participating agent types and number of agents that have to fulfill the sequence of tasks, their starting location, as well as domain specific information again, such as e. g. their provided

capacity or their speed (see Paragraph 8.1.2.3 below for the specification of a scenario).

- `coordination-mechanism`: This parameter specifies the coordination mechanism employed by the agents to solve the problem scenario, in more detail to fulfill the tasks in a self-organizing emergent manner. The available mechanisms are dependent on the chosen problem domain.

The `run settings` contain the following parameters:

- `iteration-limit`: This parameter specifies the number of iterations (time steps) that will be simulated in the run. If the limit is reached, the simulation of the run will be stopped.

- `number-of-agents`: This parameter specifies the number of (mobile) agents that will participate in the run. If the scenario configuration file specifies a higher number, this number will be decreased to the value specified here.

- `task-generation-number`: This parameter specifies the number of tasks that have to be generated randomly. These tasks may be also generated in addition to tasks specified by the scenario configuration file.

- `task-generation-limit`: If tasks are to be generated randomly, this parameter specifies the iteration (time step) in which the last task may be generated. In contrast to the iteration limit, the simulation of the run will not be stopped at the task-generation limit but proceed further until all tasks are fulfilled respectively the iteration limit is reached.

- `task-generation-probability`: If tasks are to be generated randomly, this parameter specifies the probability, by which a randomly generated task is deployed to the system. For instance, in case of a task generation probability of 0.5, a randomly generated task is deployed to the system with a probability of 50%.

The `run parameters` contain information regarding a coordination mechanism and thus have to be specified depending on the coordination mechanism specified in the run setup. In case of PIC, the run parameters specify the following parameters:

- `emission-concentration`: The concentration of an infochemical at the time of emission.

- `reward-concentration`: The concentration of a reward provided by a flower.

- `reward-concentration-variation`: In case that the concentration of the reward may vary, the factor by which the concentration varies.

- `idle-time`: In case a pollinator agent does not perceive any synomones, this parameter specifies the time after which the agent starts following hive kairomones back to the hive.
- `emission-rates`: The rate by which an agent emits infochemicals according to its specified behavior. This parameter has to be specified for every agent type participating in PIC, i. e. pollinator agents, pollenizer agents, hive agents.
- `evaporation-factor`, `threshold-concentration`, `diffusion-coefficient`: for each type of infochemical participating in PIC, i. e. pheromones, allomones, kairomones, and synomones, these three parameters specify the propagation and evaporation behavior of an infochemical.

Because the maximum number of runs is unbound, an experiment may consist of more than one run, possibly with different setups, settings, and parameters. This functionality is in particular necessary for experiments with respect to the EIA approach, in which several runs respectively run instances have to be simulated that require scenarios with different configurations regarding the tasks (multiple-scenarios typed experiments). In case that the run setup, settings, and parameters have to be the same for all runs or have to represent a default configuration for all runs, a `global` run setup, setting, and parameters can be specified.

In case that an efficiency improvement of the solution by an EIA is required, additionally some `eia` parameters have to be specified:

- `rules`: This parameter regulates if the EIA will load already existing rules for the experiment optimization from a specified file. If `load-rules` is set to true, the EIA will load the rules in run `on-run` from the file specified in `filename`.
- `output`: This parameter regulates if the statistics of the experiment will be saved to a file. If the output is enabled, the statistics will be saved to the file specified in `filename`.
- `precalculated-distances`: This parameter regulates if the calculated distances between locations of the environment are loaded or saved from or to a file. If `load-values` is set to true, the precalculated distance values will be loaded from the file specified in `filename`. if `save-values` is set to true, the distance values calculated in the experiment will be saved to the specified file. The loading of precalculated distance value will in particular accelerate the optimization process.

Logging Specification

The second part of an experiment configuration file deals with the `logging` behavior of SENSES. SENSES allows to specify different `logger`s to log various system events. A `logger` is specified by a `name`, if it has to log system events or not (`logging`), and if the logs have to be written to a corresponding log file or not (`log-to-file`).

Default Parameters Specification

The third part of an experiment configuration file deals with `default` parameters for SENSES. This part may include information on `general` parameters, such as `record`, `protocol`, and `logging` (see above), but also information on run `settings`, coordination mechanism `parameters`, or EIA-related parameters. Even a mixture of these parameter configurations can be specified.

8.1.2.2 Specification of an Environment

Listing A.2 shows the XML Schema for an environment configuration file. The specification of an environment requires an environment `name`, `width`, `height`, `scale`, and `type`. The scale defines the factor by which the environment has to be "stretched", the `type` distinguishes between `individual` environment layouts or `grid` layouts. The environment configuration further has to specify a number of `locations`, each identified by its `id` and its coordinate represented by an `x-position` and a `y-position`, as well as a number of `connections`, each identified by its `id` and its endpoints represented by a `from` location and a `to` location.

8.1.2.3 Specification of a Scenario

The specification of a scenario strongly depends on the specified problem domain defined in the run setup. In case of a PDP domain, Listing A.3 defines the XML Schema for a PDP scenario configuration file. According to the definition of a PDP (see Subsection 7.1.2), a PDP scenario configuration file consequently has to define `request-types`, `vehicle-types`, `depots`, `stations`, `vehicles`, and `requests`. A `request type` requires the specification of `sub-types` and `super-types`. The same holds for `vehicle types`, however, they additionally require the specification of linked `request types`. A `depot` is specified by its `vehicle types`, its `id`, and either a `position` or a coordinate represented by an `x-position` and a `y-position`. A `station` is specified by the `request types` that can occur at the station, its `id`, and either a `position` or a coordinate represented by an `x-position` and a `y-position`. The `type` of a station can be either `pickup`, `deliver`, or `universal`, which indicates that the station may be used for both pickup and delivery. In addition to its `id`, `position`, `x-position` and `y-position`, a `vehicle` is specified by a `capacity`, `speed`, and vehicle `type`, of course. Finally, a `request` is specified by its `pickup-station` and `delivery-station` along with the corresponding time windows specified by an `earliest` and a `latest` iteration, its `id`, `loadsize`, `service-time`, and `probability` to occur, as well as its request `type`.

8.1.2.4 Specification of a Result File

The result file of an experiment depends on the coordination mechanism as well as the problem domain. In case of PIC for PDPs, the XML Schema for a PDP result file is shown in Listing A.4. The result file specifies the `name` of the experiment and

then lists the results of every `run` of the experiment, i. e. all the measures of interest for PDPs and PIC, along with its run `number`.

8.1.2.5 Experiment Processing

If an experiment configuration file is specified as described in Subsection 8.1.2.1 and made available to SENSES as a start parameter respectively loaded from the GUI, SENSES will parse this file and generate an internal `Experiment` object. Otherwise, i. e. if the experiment is specified by means of the configurator provided by the GUI, the latter is in charge of generating the `Experiment` object. In any case, the `Experiment` object is passed to the `Experiment Manager`, which is located in the *main component* of SENSES and which is responsible for the execution of the experiment.

For the preparation of a run of the `Experiment`, the `Experiment Manager` delegates among other things the parsing of the environment configuration file (see Subsection 8.1.2.2) specified in the experiment to an `Environment Manager`, as well as the parsing of the scenario configuration file (see Subsection 8.1.2.3) to a `Scenario Manager`. The `Environment Manager` is in charge of generating `Location` objects and `Connection` objects according to the specified elements in the configuration file, the `Scenario Manager` is in turn in charge of generating `Agent`s and `Task`s, which possibly may have to be generated randomly, depending on the problem domain. At the end of the preparation of a run, the environment, agents, and tasks are ready for simulation.

When the execution of the experiment has started (see Figure 8.2), the first run is passed to the `Simulation Engine`, which then starts simulating the scenario of this run (see Figure 8.3). The simulation can be paused and resumed as well as stopped by the user, if the GUI has been started. If the simulation of a run has finished, which in the case of PDP means that all transportation requests are fulfilled and all vehicles are returned to the depot again, the next run of the experiment, if available, will be prepared as described above and then simulated as well. Otherwise, i. e. if all runs of the experiment have been simulated, the experiment will be finished and the result file (see Subsection 8.1.2.4) will be generated, if required. If the simulation of any run is stopped before, e. g. by the user, the execution of the overall experiment is stopped as well, i. e. all remaining runs will not be simulated any more. In both cases, however, the experiment can be reset to its initial state for another simulation or be cleared, which removes the `Experiment` from the `Experiment Manager`.

8.1.3 Realization of the IBC Approach

The realization of the IBC approach (see Chapter 4) in SENSES comprises in the first instance an appropriate realization of the DIC model (see Section 4.2), but also a realization of the instantiating coordination mechanisms, in this case PIC.

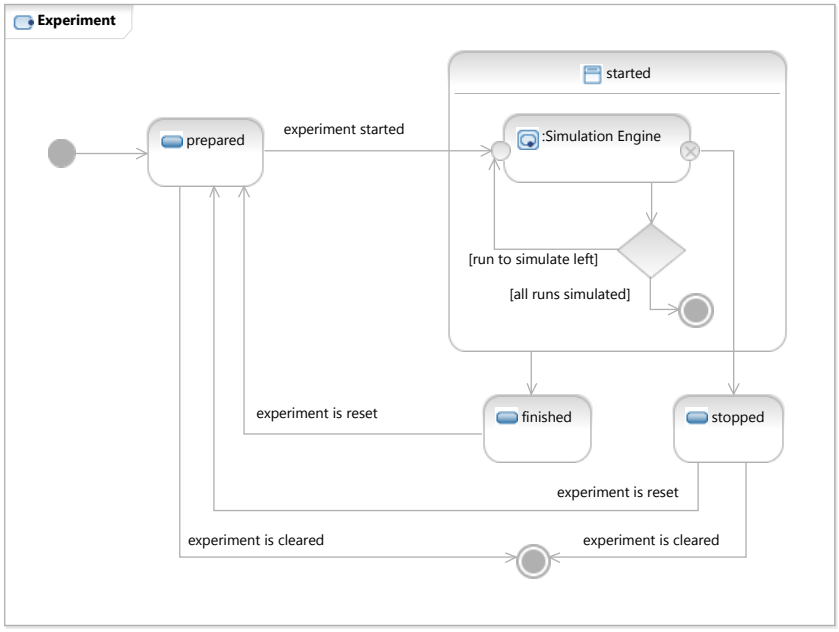


Figure 8.2: States of an Experiment in SENSES

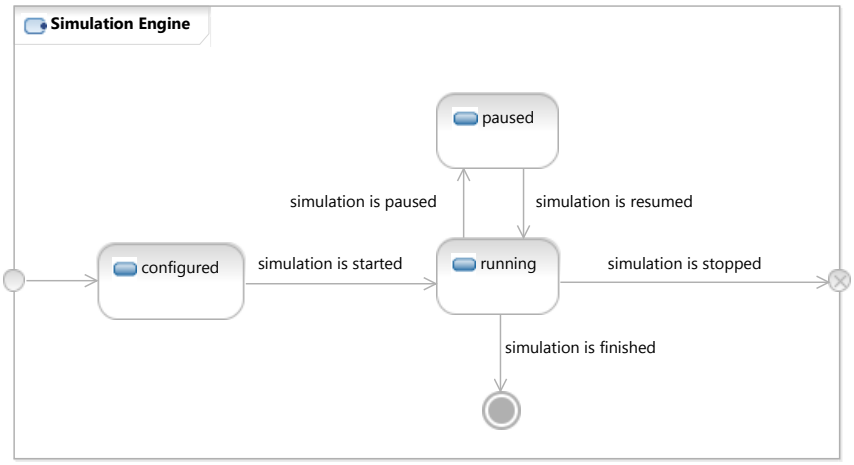


Figure 8.3: States of the Simulation Engine in SENSES

8.1.3.1 Realization of the DIC Model

The realization of the DIC model as formally defined in Section 4.2 was accomplished in the *models* component of SENSES. It turned out that a critical aspect with regard to the simulation speed and memory usage was the implementation of the infochemical propagation. According to the formal model, an infochemical emitted at a given location by an agent is propagated to neighboring locations until its current concentration falls below its threshold concentration. To simulate this propagation behavior, three alternatives are imaginable:

- **Concurrent propagation:** This alternative requires infochemicals to be realized as threads. An emitted infochemical first clones itself by the number of inbound locations at this location. Then, all cloned threads propagate "in parallel" to the neighboring locations, clone themselves again accordingly, propagate along the inbound connections at their respective locations, and so forth. If a cloned thread reaches a location already storing a similar infochemical with a higher concentration (because there is a shorter route to the initial location), the newly arriving infochemical will not be cloned any more and its propagation will stop at this location. However, the problem is that if the infochemical reaches a location already storing a similar infochemical with a lower concentration, it will replace this infochemical, because now a shorter route to the initial location has been identified. Unfortunately, the prior thread has already cloned itself so that now the routing information at all following locations has to be updated again by cloning and propagating the actual thread with the higher concentration and so forth. This may result in a high amount of unnecessarily created threads, which makes this alternative not very appropriate.

- **Depth-first propagation:** In this alternative, an infochemical is realized as a regular object. Similar to a depth-first search in a tree, the infochemical is first cloned for one of the inbound connections at its initial location and then propagated over this connection to the neighboring location. There, the infochemical again is cloned for one of the inbound connections and so forth. If a propagation is not possible any more, the infochemical is cloned and propagated over a second inbound connection at the last location and so forth. Apparently, this alternative suffers from the same problem as the alternative described above with regard to the unnecessarily created objects. Moreover, this alternative in addition takes much more simulation time because the propagation is not parallelized, which makes this alternative not very appropriate as well.

- **Breadth-first propagation:** This last alternative turned out to be the best with regard to simulation speed and memory usage. Thereby, an infochemical, realized as a regular object as well, first is cloned at its initial location by the number of inbound connections of this location. Similar to a breadth-first

search in a tree, every cloned infochemical is then propagated only over the respective inbound connection. If the propagation was successful, i. e. if no similar infochemical with a higher concentration has been already stored at the neighboring location, the infochemical is not only stored at its destination but its reference is also put into a waiting queue at the initial location. After all cloned infochemicals have been propagated to the neighboring locations of the initial location, the first infochemical contained in the waiting queue is removed and then cloned at its respective location according to the number of inbound locations of this location. If a propagation of such a cloned infochemical has been successful, its reference is similarly put at the end of the waiting queue at the initial location. In case that any location has already stored a similar infochemical with a lower concentration, the infochemical will not only be removed from the infochemical buffer, but the reference of this infochemical will be also removed from the waiting queue, if necessary, so that it will not be propagated any further. Thus, the number of unnecessarily created objects remains as low as possible. This propagation process continues until the waiting queue is empty.

Another critical aspect with regard to simulation speed and memory usage has been the number of infochemical instances that had to be created and stored in the memory. Because larger scenarios require the simulation of some millions of infochemical instances, we have realized a central pool of infochemicals that functions as a resource pool and stores any created infochemical instance. If the instance is no longer used by the simulation (e. g. in case of evaporation), it thus can be reused, which saves simulation time for object creation and garbage collection.

8.1.3.2 Realization of the PIC Mechanism

The realization of the PIC mechanism as formally defined in Section 7.2 was accomplished in the *mechanisms* component of SENSES. According to the specification, the PIC mechanism requires a couple of "active" elements and agents, including the locations, the hive agents, the pollenizer agents, and the pollinator agents. In many agent-based simulation tools, these elements and agents are realized as separate threads to conform to the used paradigm. However, for evaluation purposes it has to be guaranteed that in every iteration any element or agent receives enough processing time to accomplish its actions. In general, this requirement is not fulfilled for threads, which is why we have realized these simulation elements as "regular" objects.

An important aspect with regard to the solution performance then is the order in which these simulation elements receive processing time in an iteration. A bad order might worsen the overall solution, because some simulation elements might not have the necessary information for their actions only because this information is generated some time later in this iteration. The order in which the simulation elements receive processing time in each iteration is thus realized as follows:

1. **Locations:** Every location evaporates all infochemicals stored at its outbound connections and, if the current concentration of an infochemicals is below its threshold concentration, removes the infochemical.

2. **Hive agents:** Every hive agent emits kairomones that guide pollinator agents on their way back to this hive. All kairomones are propagated within one single iteration.

3. **Pollenizer agents:** Every pollenizer agent will emit synomones, if it provides or demands pollen grains. It will emit allomones, if a pollinator just fulfilled its demand, respectively. Synomones and allomones respectively are propagated within one single iteration.

4. **Pollinator agents:** If a pollinator agent is currently situated on a location, it will perceive all infochemicals stored at its current location, calculate the utility of all synomones, and move along the outbound connection storing the synomone with the highest utility, while emitting pheromones. The pheromones are propagated within one single iteration. If a pollinator agent is currently located on a connection, it will further move along the connection. If a pollinator is situated on a location with an appropriate pollenizer agent, it will start respectively continue the exchange of pollen grains.

8.1.4 Realization of the EIA Approach

The realization of the EIA approach (see Chapter 6) focuses mainly on the realization of the instantiation of the EIA customized for PDPs (see Section 7.3). The focus in this section is mainly on the reception of the local agent histories and their transformation to a global system history, along with the implementation of the agent advisement.

8.1.4.1 Receive Local Agent Histories

As already mentioned in Subsection 7.3.4.1, two different types of data are of interest for this action: environment data and agent data. In the given realization, the environment data is collected by the EIA at the beginning of each run. Therefore, the EIA is able to directly access the map of the environment used by the basic MAS in SENSES. Technically, an *aspect* [FECA04], which is part of the EIA realization, is used to tap into the `Scenario Manager`, more precisely the method that creates the map according to the information specified in the scenario configuration file that is parsed (see Subsection 8.1.2.3). Thus, the EIA receives a complete view of the environment structure. In turn, the agents of the basic MAS do not explicitly have to collect data about the environment in their local histories, which brings along the advantage that the agents do not have to be augmented with such an additional functionality and appropriate data structure. Admittedly, this is only possible in systems that do allow accessing the agents data structures by an aspect directly.

Because the representation of locations and connections for the realization of the PIC mechanism is quite complex, as e. g. locations handle the evaporation, propagation, and aggregation of infochemicals, the representation of locations and connections in the internal data structure of the EIA is simplified. Therefore, the entire original graph is transformed by the EIA into a simpler form by traversing the graph and creating `Simplified Location`s and `Simplfied Connection`s, respectively. If this operation is performed repeatedly on all locations created by SENSES and the result of each operation is added to the current result, a complete graph will be created. To avoid unnecessary work, only locations not touched before are transformed. After this operation the graph is represented by a list of `Simplified Location`s. To calculate shortest routes in the resulting graph, an A*-algorithm [HNR68] is employed.

In order to collect the data about the agents, technically again an *aspect* as part of the EIA realization is used to record these actions instead of using a log file for each agent, which results in the same advantage as described above. For each agent, every basic action of interest is recorded in an `AgentAction`, along with the iteration and the location at which they appeared, as well as the infochemical that the agent was following at that point in time:

```
AgentAction = (Iteration, Location, Infochemical, AgentActionType)
```

According to the list of basic actions of interest (see Subsection 7.3.4.1), the parameter `AgentActionType` hence can be one out of of the following possible values: `CHANGE_INFOCHEMICAL`, `SERVICE_STATION`, or `MOVE_TO_DEPOT`. `Location` and `Infochemical` are the respective references to the PIC mechanism realization.

An element of a local agent history thus consists of a list of `AgentAction`s ordered by the iteration in which the action was performed. The action *receive* adds this list to the internal data structure of the EIA, which is essentially a map where the respective agent is the key and a list of the agent actions is the value. At this point, the data structure for the agent is the original agent system's representation with no simplification.

8.1.4.2 Transform Local Agent Histories into Global History

The global history, which is created by the action *transform*, is represented as a container, named `SimulationDataContainer`, that subsumes all relevant global information of a single run as well as information about the map and the location of the depot (see Figure 8.4). The collected local agent histories are transformed into `Task`s and `Vehicle`s. A `Vehicle` represents one of the basic agents fulfilling the `Task`s and consists of an identifier, its capacity, and the initial location where the vehicle was located at the beginning of the run. A `Task` is the minimal unit of work, i. e. a pickup and a delivery. It has an identifier, the number of goods that have to be transported (loadsize), timestamps that show when the task became valid and when the time to perform the task was over, as well as the locations of the pickup station

and the delivery station. A `TaskVehicleAssignment` contains information about which vehicle performed which task and the order in which the task was performed in the system. To simplify the optimization process, the `SimulationDataContainer` only contains identifiers for tasks and vehicles instead of references to them.

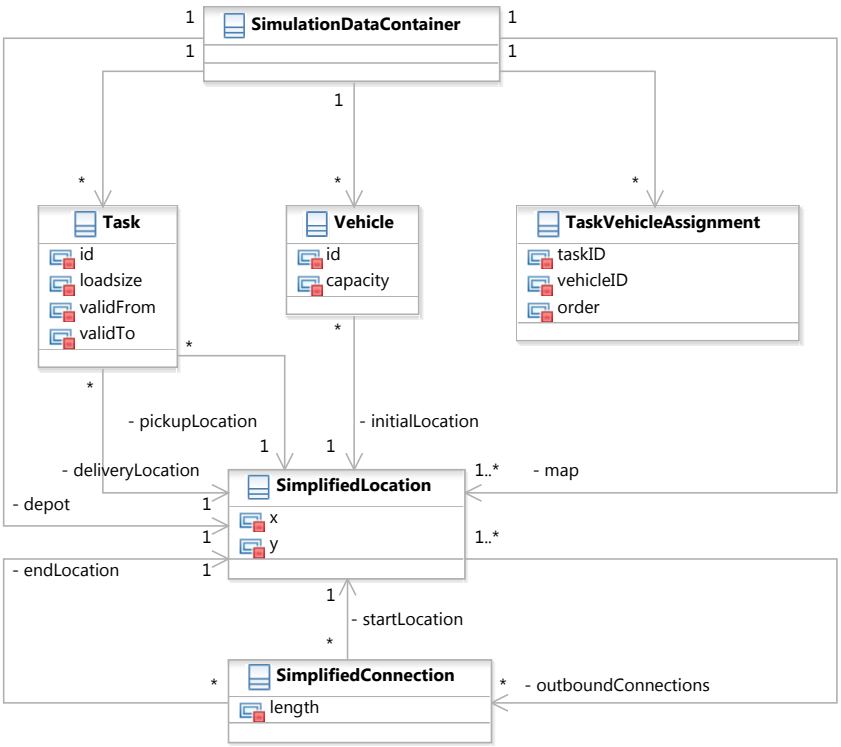


Figure 8.4: Representation of a global history in the EIA realization

Technically, the transformation of agent data is a simple rule based process. As mentioned above, there exists a list of `AgentActions` ordered by the iteration for each agent. Applying rules to this list allows for the derivation of several types of data of interest. To derive that an agent has serviced a task, the following conditions must hold:

```
Agent A at Iteration x1 at Location l1 followed Infochemical i1
        at Iteration x2 at Location l2 followed Infochemical null
        at Iteration x2 at Location l2 serviced Station
        at Iteration x3 at Location l2 followed Infochemical i2
        at Iteration x4 at Location l3 followed Infochemical null
        at Iteration x4 at Location l3 serviced Station
        i1 instanceof PollenizerSynomone
        i2 instanceof PollenizerSynomone
```

If all of the above conditions are true, a `Task` can be constructed as follows:

```
Task : pickupLocation = l2
       deliveryLocation = l3
       load = i1.numberOfProvidedPollen
       validFrom = i1.flower.bloomingBegin
       validTo = i2.flower.bloomingEnd
       id = new Id
```

Each infochemical contains attributes for the provided load or for the supply respectively demand (`numberOfProvidedPollen` and `numberOfDesiredPollen` respectively). Additionally, an infochemical carries a time stamp `bloomBegin` that indicates when the pollenizer agent, in more detail the flower, first emitted an infochemical that described this transportation request. The flower itself is referenced in the field `flower` and contains an attribute `bloomEnd` that indicates the iteration at which the agent wants the request to be performed at the latest. By using the value of the agent to which the goods have to be transported, it is ensured that this value really shows the end of the acceptable time window for the entire task.

If it is the first time an agent `a` has been encountered by the EIA, a `Vehicle` is instantiated as well:

```
Vehicle: initialLocation = l1
         capacity = a.capacity
         id = new Id
         basicId = a.pollinatorId
```

The `Vehicle` contains two identifiers: One (`id`) that is unique in the EIA and is used e. g. to locate the vehicle in internal arrays and one (`basicID`) that is used to identify the agent in the basic system. Please note that the assignments of the location variables really imply a conversion between the original environment model used by the basic system and the one used by the EIA.

Furthermore, the EIA can infer that the vehicle serviced the task. Therefore, a `TaskVehicleAssignment` is created. As in Definition 6.3, this construct also contains an additional parameter $t \in \mathbb{N}$ that designates the order in which the tasks were performed within the system based on the iteration when an agent arrived at the pickup location of the task. This additional variable allows to order the assignments according to the times they were executed by the agents.

Because only the global history is used by the further actions, the local agent histories are only relevant until the global history has been extracted from it. Therefore, the given realization does not save the local agent histories after the transformation has been performed.

8.1.4.3 Advising the Agents of the Basic System

In order to influence the local behavior of the agents of the basic system by ignore rules, we implemented a mechanism to make the agents aware of the advice created by the EIA and to incorporate it in their decision process. Therefore, we augmented

the pollinator agents by a rule engine that applies rules within the decision function of the pollinators (see Definition 6.7). At the beginning of each simulation run, the rules are made available in the agents situational data and registered with the agent's rule engine. From a technical point of view, an aspect is used again. It instruments the constructor of a `PollinatorAgent` and sets the list of rules.

When a pollinator agent evaluates the utility of the perceived synomones at its current location, the rule engine compares each synomone to the abstracted synomone that is part of the rule. If the synomone is similar to the abstracted synomone, i. e. $dist_{syn} < synthresh$, the rule is applied by changing the utility of the synomone by multiplying the utility with a factor that is part of the rule's action a:

$$u(\varsigma)' = \prod^{Rule'} a(r) \cdot u(\varsigma)$$

where $Rule' \subseteq dat_{EIA}^{Rule_i}$ and $\forall r \in Rule' : eval(r) = true$.

Effectively, this changes the action the agent chooses to perform. If several rules match the infochemical, all of them are applied, i. e. all factors are multiplied with the utility.

8.2 Case Studies: PDPs in Intralogistics

Whereas the last section has described the simulation tool used to execute experiments, this section describes two case studies from the PDP domain, which provide the basis for the experiments. Even though PDPs can be found in various areas (see Section 7.1), the primary application area of PDPs is without doubts logistics. However, due to the long history of logistics as well as different products, companies, and systems involved in the processes, there exists no true or unique definition of logistics [RCB06]. A very general approach to describe logistics is provided by the seven-rights-definition (cf. [Plo64]): logistics means to deliver the right product, in the right quantity and the right condition, to the right place at the right time for the right customer at the right price.

However, when combining the terms PDP and logistics, most people immediately will think of external logistics, i. e. trucking companies transporting goods between different landmarks. But PDPs also appear in the growing field of internal logistics, which has similarly a high potential for cost-reduction and savings. Internal logistics, which is also referred to as intralogistics or in-house material handling, describes the internal flow of materials as well as the corresponding flow of information between different logistics points inside a company. Internal logistics is present in many companies of various industries, such as automotive, print, chemical, hospital, food and beverage, newspaper, pharmaceutical, paper, manufacturing, or warehouses and distribution centers. Hence, in-house material handling may stretch from basic or raw materials over work in progress to complete products.

In contrast to the transportation of goods between companies by trucks with drivers, the transportation of goods within a company today is more and more

accomplished by so-called automated guided vehicle (AGV) systems [Vis06]. An AGV system in general consists of several parts, namely the AGVs, a transportation network, physical interfaces between the storage or production system and the transportation system, as well as the control system, which typically contains a vehicle manager (central server) connected to the management system of the company. AGVs are custom-made, unmanned vehicles able to transport different kinds of goods. There exist various types of AGVs, e. g. forked AGVs as typically present in warehouses and distribution center, unit load AGVs as typically present in manufactures, or tug/tow AGVs as well as more specialized AGVs dependent on the customers' needs. In addition to floor pickups and drops, AGVs can be designed to interface with other (types of) AGVs as well as many types of stationary equipment, including conveyors, racking, machines, or stands. At these stations, pickup and delivery points are installed that operate as interfaces between the storage or production system and the AGVs. At these points, goods are transferred to or from an AGV. The transportation network connects all stationary equipments. The AGVs can move on this transportation network, guided by a laser navigation system or following a physical path on the factory floor that is marked by magnets or cables that are fixed in the floor. AGVs use batteries as energy source, which are typically charged at opportunity at dedicated zones.

Today, transportation is either initiated by a human operator that scans goods for identification and formulates a transportation request, or the requests are initiated automatically by sensors or supervisory computer systems. A transportation request is traditionally communicated to the vehicle manager of the AGV system that is then responsible for selecting an available AGV to carry out the transport request based on a predefined assignment strategy, e. g. *nearest vehicle strategy*, *longest idle vehicle strategy*, etc. (see [Vis06] for an overview). The vehicle manager communicates the request to a PC-based on-board vehicle controller, which lets the AGV move to the appropriate pickup station. After arrival and pickup, the AGV receives its destination. The AGVs constantly communicate their location, their battery status, their current destination, and whether they are loaded or empty to the central vehicle manager or the latter polls this data from the AGVs itself.

Consequently, a vehicle manager is in charge of numerous complex and time-consuming tasks, such as transportation request management, vehicle routing, collision avoidance, deadlock avoidance, and system control, whereas the AGVs have only a limited degree of autonomy. In recent years, this centralized architecture has been successfully deployed in numerous practical installations yielding many advantages such as efficiency, configurability, and predictability. However, especially in highly dynamic environments, where the situation changes frequently and numerous AGVs are engaged, problems with this centralized architecture have been experienced (cf. [Vis06]). This fact together with the evolution of the market puts forward new requirements for AGV systems (cf. [WSHL05]):

- AGVs should be able to exploit opportunities, e. g. when an AGV is assigned a transportation request and moves toward the pickup station, it should be

possible for this AGV to switch tasks along the way if a more interesting transport request pops up.

- AGVs should be able to cope with particular situations, e. g. when new goods arrive in the environment, the AGVs should be able to reorganize themselves smoothly.

- The AGV system should be able to deal with AGVs leaving the system, e. g. for maintenance or charging their batteries, as well as AGVs (re-)entering the system.

- AGVs should also be able to anticipate possible difficulties themselves, e. g. when the battery level of an AGV decreases.

To summarize, modern in-house material handling in future requires a higher flexibility and openness of AGV systems, which necessarily requires decentralized solutions. In the following, we exemplary describe two case studies from the field of internal logistics in more detail, to whom we have applied our developed concepts for evaluation purposes (see Section 8.3 and 8.4).

8.2.1 Tire Warehouse

The first case study covers in-house material handling in a tire warehouse. In such a warehouse, tires constantly arrive from the manufacturing system in the – usually directly affiliated – *receiving area* of the tire warehouse and have to be transported by AGVs to dedicated locations in the *storage area*. In order to maximize the use of the available space, most tire warehouses utilize narrow aisles with high racking or block stacking to store the tires. Some time later the tires again have to be picked up by AGVs at the storage area and have to be delivered to dedicated floor positions in the *shipping area*, in preparation for shipment. Finally, the tires have to be moved from the floor positions in the shipping area into different trailers for customer delivery, which can be accomplished by AGVs or human workers by means of sack trucks or forklifts as well.

For the evaluation, however, we only consider the transportations between the storage and the shipping area of the tire warehouse. The transportation of tires between the receiving and the storage area as well as between the shipping area and the trailers are masked out, because conceptually there is no difference compared to the transportations between the storage and shipping area, it only makes the evaluation more confusing. We also presume that the tire manufacturer at this site only produces tires for a certain type of vehicle, e. g. only for cars but not for buses or trucks as well. This means that all AGVs are able to transport all types of tires in the warehouse, which requires only AGVs of one single type. However, tires may still vary in their rubber compound, width, diameter, tread, etc.

8.2.2 Automotive Manufacturing

The second case study covers in-house material handling in automotive manufacturing. Advanced and adaptive assembly of cars will be essential to reduce production costs to a minimum and to yield higher levels of energy efficiency across the entire manufacturing process. According to the European Council for Automotive R&D (EUCAR), the identification and development of smart and flexible manufacturing processes will be fundamental to ensure the competitiveness of the European automobile industry [EUC09]. This requires flexible manufacturing systems, which guarantee performance and robustness despite highly variable production volumes, flexible assembly equipment to easily adapt to market variations, and highly reconfigurable assembly operations in order to enhance production flexibility. The vision of such self-adaptive plants that accommodate diversified customer needs is also pursued within the EU Integrated Project 'MyCar' [EU 10].

The production of a car in general starts in the press shop, where components such as roofs, doors, side frames, front and rear lids, etc. are built out of sheet metal, proceeds to the bodyshop, where up to 500 stamped parts are welded, glued, riveted, and bolted to form the body of the car, up to the paintshop, where painting machines apply several layers of paint on to the car body. This painted body is then queued into an assembly line, where pre-assembled components such as the engine, axles, doors, bumpers, as well as the interior such as the dashboard or seats are assembled just in sequence. After a final quality check the produced car leaves the plant for shipping. During these processes, up to 20.000 individual parts are assembled to form one single car. In this ways, for example, the BMW production plant in Dingolfing produces more than 1.200 cars of the 5, 6, and 7 Series every day[2].

For the purpose of this thesis we are not interested in the entire manufacturing process but only in one single section, namely the transportation of certain stamped parts from the press shop to the bodyshop. In contrast to the first case study, this second case study therefore involves five different types of parts: roofs, doors, side frames, floor plates, as well as front and rear lids. Each type of parts is delivered from the press shop bundled in racks, each consisting of a certain amount of parts, whereby the amount depends on the size of the parts. These racks have to be transported from central gates by AGVs to different assembly lines, each comprising five delivery points, i. e. one delivery point for one specific type of parts at a time. Because the racks have different dimensions depending on the included parts, in this second case study three different types of AGVs are required for transportation: one AGV type can handle racks with roofs or lids, one type can handle racks with doors or side frames, and one type can handle racks with floor plates. Thus, the amount of parts in each rack is of no importance to an AGV, but only the dimension of a rack. Usually, an AGV of any type can only transport one rack at the same time.

[2]Figures taken from http://www.bmw-werk-dingolfing.de

8.3 Experiments Regarding the IBC Approach

Based on the two case studies described in the previous section, this section documents the experimental evaluation that we have performed regarding the IBC approach, i. e. the application of the PIC mechanism to these PDP case studies. This includes the description of the experiment preparation and execution as well as the presentation and analysis of the corresponding experimental results. The hypothesis, which has to be proven by the experiments, is that the coordination by means of different types of infochemicals, i. e. infochemicals with different functions, dynamics, and semantics as provided by the DIC model, will yield both more efficient solutions to PDPs as well as more efficient solution processes compared to the coordination by means of only one single type of infochemicals, as e. g. provided by pheromone-based coordination. The experimental evaluation of the EIA approach is described in the next section.

Unfortunately, we cannot provide any benchmarking data in comparison to existing solution methods by the OR community. This has several reasons: (1) Appropriate benchmark instances for such dynamic PDPs do not exist. (2) Possibly existing solution methods by the OR community for PDPs of such a huge size and complexity as we have experimented with, are publicly not available. (3) We have specified experiments with more realistic constraints than specified by the OR community. In particular, we do not assume that only unloading goods at the delivery station requires time, but also the loading of goods at the pickup stations. We furthermore do not assume that there always exists a direct connection between two stations, but allow for junctions and intermediate points. Moreover, we do not assume that one pickup/delivery station occurs only once in a PDP, but allow for multiple occurrences of stations.

8.3.1 Experiment Preparation

To test and evaluate the capabilities of the IBC approach respectively the PIC mechanism, we have created two kinds of scenarios:

- **Scenarios requiring one type of AGVs:** The first kind of scenarios, which are based on the tire warehouse case study, only require one single type of AGVs for the solution. We will use four different environments of different size, which due to the case study exhibit a quite symmetrical layout.

- **Scenarios requiring multiple types of AGVs:** The second kind of scenarios, which are based on the automotive manufacturing case study, require multiple types of AGVs for the solution. Here, we only use one type of environment, which, however, exhibits a quite unsymmetrical layout.

For the purpose of evaluation, in both kinds of scenarios we have performed five different series of experiments based on the PIC mechanism:

1. **Coordination by means of synomones (S):** In this experiment series, the coordination between pickup/delivery stations and AGVs is accomplished by means of synomones only, i. e. the agents are not able to emit allomones or pheromones. In other words, the coordination between the agents takes place based on infochemicals without different functions and semantics. This series consequently represents the baseline for all further improvements.

2. **Coordination by means of synomones and allomones (SA):** In this experiment series, the basic coordination by synomones is additionally extended by allomones emitted by pickup stations to keep further AGVs off from visiting.

3. **Coordination by means of synomones and pheromones (SP):** In this experiment series, the basic coordination by synomones is in contrast additionally extended by pheromones emitted by AGVs to distract the attention of subsequent AGVs from the intended target pickup station.

4. **Coordination by means of synomones, allomones, and pheromones with general parameter settings (SAPg):** In this experiment series, synomones, allomones, and pheromones are used for the coordination, however, all types of infochemicals are configured with the same parameter settings, i. e. they have the same dynamics.

5. **Coordination by means of synomones, allomones, and pheromones with individual parameter settings (SAPi):** In this experiment series, synomones, allomones, and pheromones are used as well for the coordination, however in contrast, every type of infochemical is configured individually. This experiment series utilizes the full potential of the DIC model having infochemicals with different functions, semantics, and dynamics.

Please note that kairomones emitted by a depot are used in every experiment series, as otherwise AGVs would not be able to return to the depot. However, as the use of kairomones has the same effects on the efficiency in every experiment series, we have not mentioned them in every series again. If the environment map is made available to every AGV prior to the solution process, kairomones could be replaced by an appropriate A*-algorithm or any other routing algorithm. In the following, we explain the created scenarios in more detail, starting with the scenarios requiring one type of AGVs and afterwards scenarios requiring multiple types of AGVs.

8.3.1.1 Scenarios Requiring One Type of AGVs

As mentioned above, the scenarios requiring one type of AGVs are performed on four tire warehouse environments of different size (*TW-Env-1* to *TW-Env-4*, see Figures 8.5 to 8.8). Figure 8.5 exemplary illustrates such an environment. The thicker black dots correspond to the considered pickup/delivery points in the warehouse, i. e. high racks in the storage area, floor positions in the shipping area, and the depot of

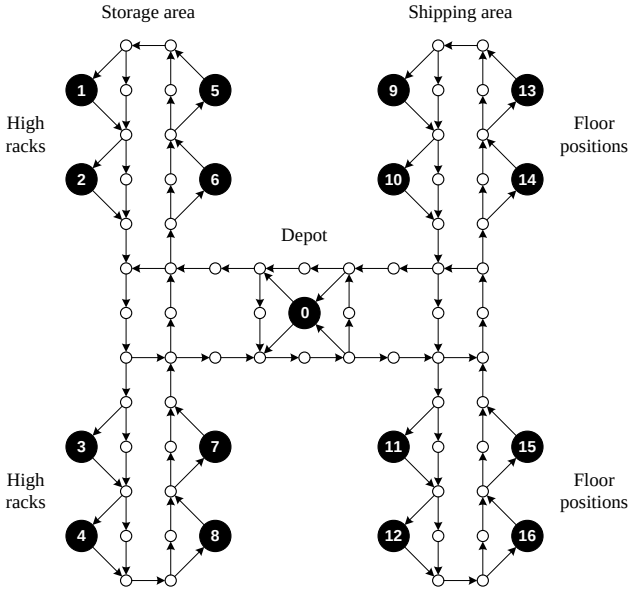


Figure 8.5: Tire warehouse environment *TW-Env 1*

the AGVs, respectively. The black circles connected by directed arrows correspond to the transportation network. Each circle thereby corresponds to a junction or another relevant network point. The direction of an arrow indicates the possible driving direction for an AGV for the respective connection. The depot, with ID=0, at which idle AGVs may charge their batteries and/or wait for new transportation requests, is located in the middle of the environment. Stations 1–8 on the left side of the depot represent high racks in the storage area, i. e. the pickup stations, whereas stations 9–16 on the right side of the depot represent floor positions in the shipping area, i. e. the delivery stations.

Figures 8.6 to 8.8 depict the three additional tire warehouse environments. The structure of these three environments remains the same, i. e. the depot is always located in the middle of the environment, whereas the pickup stations in the storage area are located at the left side and the delivery locations in the shipping area are located at the right side. The environments however all vary in their amount of locations and connections. Table 8.1 presents a more detailed overview on these numbers. The table illustrates that the locations as well as the connections nearly reduplicate in every environment.

Table 8.2 gives an overview of the different created scenarios performed on these environments as well as the other properties of these scenarios. On every environment, scenarios with three different amounts of randomly generated tasks are performed, starting from 100 tasks over 250 tasks up to 500 tasks. The number of pickup and delivery stations in the scenarios varies according to the size of the respective environment. In the smallest scenarios, 8 pickup stations and 8 delivery stations are used, whereas in the biggest scenarios 64 pickup and 64 delivery stations

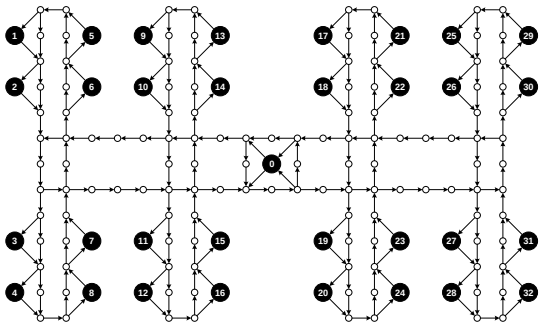


Figure 8.6: Tire warehouse environment *TW-Env 2*

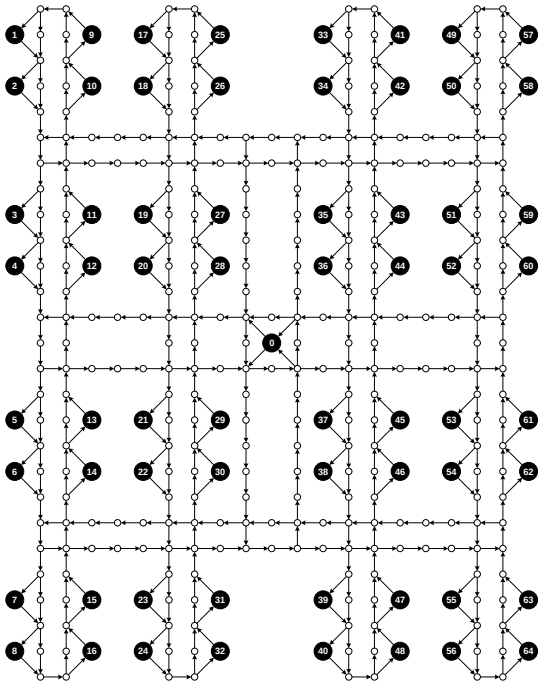


Figure 8.7: Tire warehouse environment *TW-Env 3*

Environment	Locations	Connections
TW-Env 1	81	108
TW-Env 2	161	212
TW-Env 3	369	488
TW-Env 4	721	952

Table 8.1: Environment sizes for scenarios requiring one type of AGVs

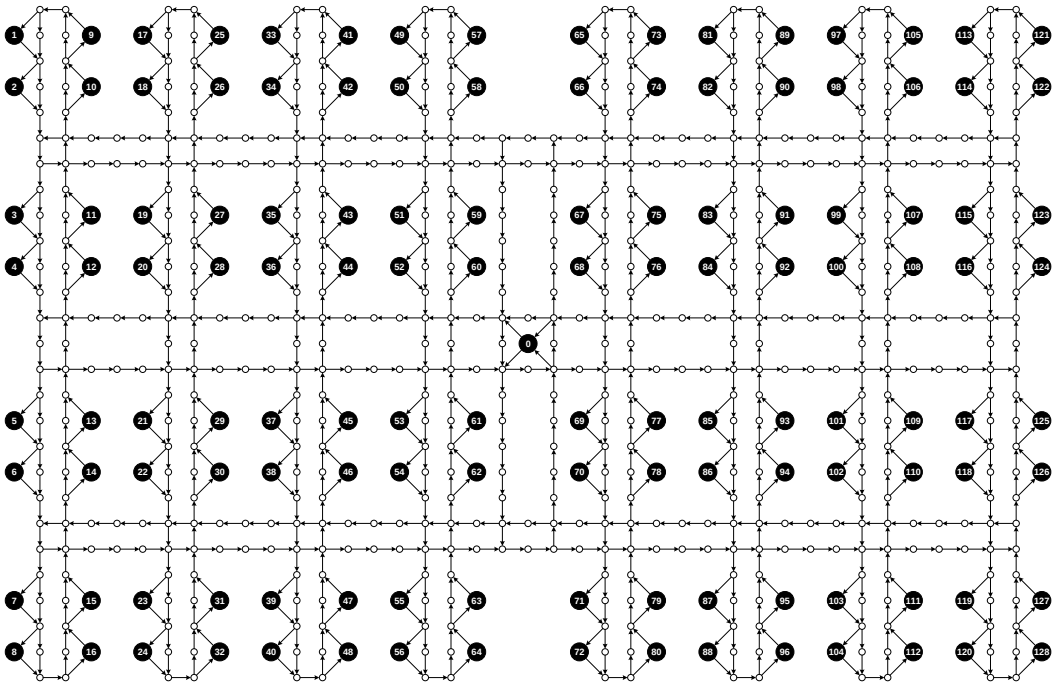


Figure 8.8: Tire warehouse environment *TW-Env 4*

are used. The number of AGVs used in every scenario varies from 10 to 100. The table reveals that these scenarios also allow for the evaluation of the scalability of the PIC mechanism in three different dimensions: (1) the number of tasks, (2) the number of pickup and delivery stations, and (3) the number of AGVs.

All parameters of a task are generated randomly, more specifically the pickup station and pickup time, the delivery station, and the loadsize. The latter varies randomly between 1 and 20 tires. An AGV may transport up to 20 tires at the same time, so that each request can be fulfilled by a single AGV, although an AGV may fulfill more than one request at the same time. All requests are available to the agents within the first 500 iterations. As a consequence, whereas in the scenarios with 100 tasks a request appears only every fifth iteration on average, in the scenarios with 500 tasks a request appears in every iteration on average. The *degree of dynamism* (see Subsection 7.1.2) of all PDP scenarios is 100%, i. e. we only experimented with highly dynamic scenarios with no static requests known in advance. The *effective degree of dynamism* of all PDP scenarios is in terms of figures 50%, where the planning horizon is assumed to be 500 iterations. Due to these figures, the PDP considered here is categorized according to [BCGL07] as a [1-1 |P/D |-] problem respectively dynamic, less-than-truck-load, multi-vehicle PDP (see Subsection 7.1.1).

Scenario	Tasks	Stations		Depot	AGVs	Environment
		pickup	delivery			
tw-env1-100	100	8	8	1	10 – 100	TW-Env 1
tw-env1-250	250	8	8	1	10 – 100	TW-Env 1
tw-env1-500	500	8	8	1	10 – 100	TW-Env 1
tw-env2-100	100	16	16	1	10 – 100	TW-Env 2
tw-env2-250	250	16	16	1	10 – 100	TW-Env 2
tw-env2-500	500	16	16	1	10 – 100	TW-Env 2
tw-env3-100	100	32	32	1	10 – 100	TW-Env 3
tw-env3-250	250	32	32	1	10 – 100	TW-Env 3
tw-env3-500	500	32	32	1	10 – 100	TW-Env 3
tw-env4-100	100	64	64	1	10 – 100	TW-Env 4
tw-env4-250	250	64	64	1	10 – 100	TW-Env 4
tw-env4-500	500	64	64	1	10 – 100	TW-Env 4

Table 8.2: Properties of tire warehouse scenarios

8.3.1.2 Scenarios Requiring Mulitple Types of AGVs

In contrast to the previous scenarios, the following scenarios focus on the transportation of different types of goods by different types of AGVs. Thus, in these scenarios we only use one environment layout of an automotive manufacturing plant, which is shown in Figure 8.9. The size of this environment in terms of locations and connections is in between *TW-Env 2* and *TW-Env 3*.

Due to the required three types of AGVs, in this environment, there are three depots with ID 0, -1, and -2, even though all AGVs in general could be housed in one single depot as well. Stations 1–5 are the receiving gates, i. e. the pickup stations, where the racks with the welded or stamped parts arrive (either from third-party vendors or more usually from the welding and stamping machines). The gates are bound to a specific type of rack. Station 1 is the dedicated pickup station for racks with front doors, station 2 is dedicated to racks with back doors, station 3 is dedicated to racks with fenders, station 4 is dedicated to racks with hoods, and station 5 is dedicated to racks with trunk lids. There are four assembly lines composed of the stations 6–10, 11–15, 16–20, and 21–25, i. e. the delivery stations. These stations are bound to a specific type of rack as well. The first station of each assembly line (the one with the lowest number of an assembly line) is always the dedicated delivery station for racks with front doors, the second station for racks with back doors, the third for racks with fenders, the fourth for racks with hoods, and the fifth for racks with trunk lids. The capacity of every AGV is 1, so an AGV may only fulfill one transportation request at the same time.

Table 8.3 gives an overview of the different created scenarios performed on the environment as well as the other properties of these scenarios. Here, scenarios with

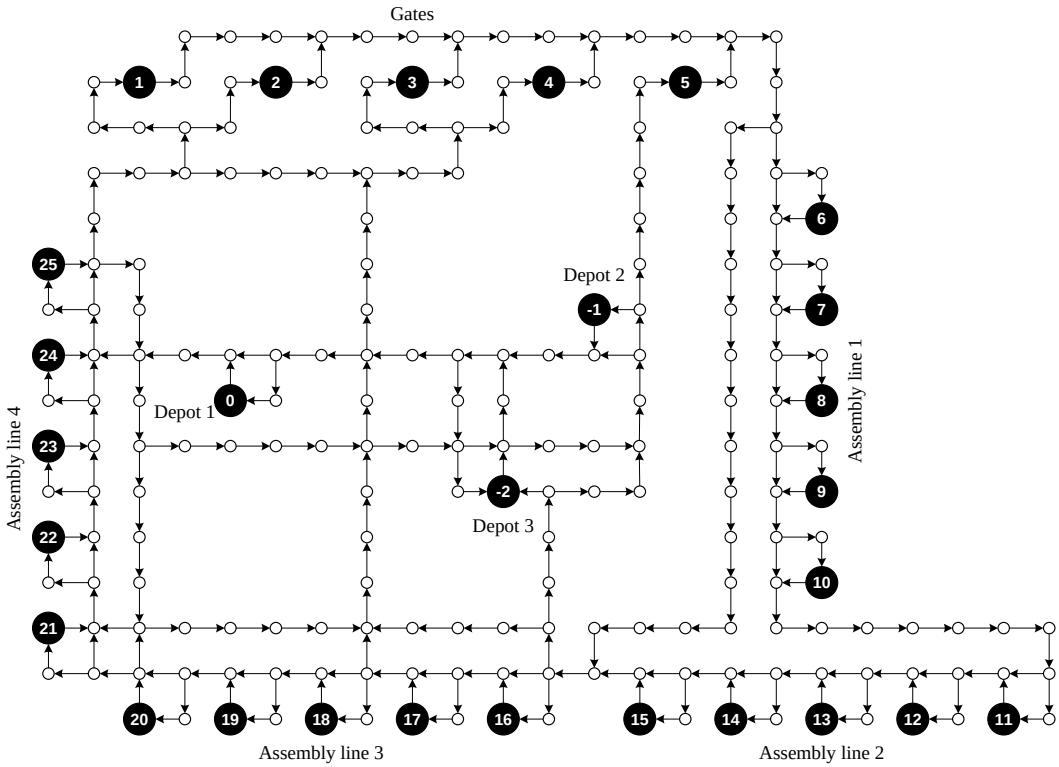


Figure 8.9: Automotive manufacturing plant environment *AM-Env*

three different amounts of randomly generated tasks are performed as well, again starting from 100 tasks over 250 tasks up to 500 tasks. The number of pickup and delivery stations in the scenarios remains the same, having 5 pickup stations and 20 delivery stations. The number of AGVs used in every scenario varies again from 10 to 100. However, because these scenarios require multiple types of AGVs, the distribution of available AGVs on these types was nearly balanced, i. e. the difference between the size of all groups of agents varies by 1 at a maximum. Similarly, the distribution of transportation requests on the available types has been nearly balanced as well.

Scenario	Tasks	Stations		Depots	AGVs	Environment
		pickup	delivery			
am-env-100	100	5	20	3	10 – 100	AM-Env
am-env-250	250	5	20	3	10 – 100	AM-Env
am-env-500	500	5	20	3	10 – 100	AM-Env

Table 8.3: Properties of automotive manufacturing scenarios

Again, the parameters of a task are generated randomly, i. e. for this case study the pickup station and pickup time as well as the delivery station. The loadsize is set to 1 rack. An AGV may transport 1 rack at the same time, so that each request can be be fulfilled by a single AGV such that an AGV may fulfill only one request at the same time. All requests are available to the agents within the first 500 iterations. The *degree of dynamism* (see Subsection refsec:ProblemDefinition) of all PDP scenarios is 100% again. The *effective degree of dynamism* of all PDP scenarios is in terms of figures 50%, where the planning horizon is assumed to be 500 iterations. Due to these figures, the PDP considered here is categorized according to [BCGL07] as a [1-1 |P/D |-] problem respectively dynamic, full-truck-load, multi-vehicle PDP (see Subsection 7.1.1).

8.3.2 Experiment Execution

To evaluate the solutions to the above PDP scenarios, we were interested in different measures, which are usually of interest with regard to AGV systems (cf. [Vis06]):

- **Total travel costs (TTC):** This measure represents the total distance of all AGV movements and is an essential factor for the operational expenditures of an AGV system. Longer routes require a higher energy consumption by the AGVs as well as higher maintenance costs. The minimization of TTC thus is important to reduce the TCO of these systems.
- **Completion time (CT):** This measure represents the time required to complete all tasks, i. e. the makespan. This measure is in particular important for AGV system operators, as a smaller CT e. g. helps to achieve higher production rates. It is also required for the determination of the solution efficiency,

as the TTC may be reduced on the costs of the CT, whereas the CT can be increased on the costs of TTC.

- **Throughput (TP):** Similar to the CT, the throughput of an AGV system, i. e. number of goods handled per iteration, is very essential e. g. for the production rate of an AGV system operator. As higher the throughput, as better the solution.
- **Total rate of infochemicals (TRI):** This measure represents the amount of infochemical objects that have to be generated for the coordination process, i. e. the coordination costs. The TRI is in particular required for the determination of the solution process efficiency, as a solution may be improved on the costs of the communication, whereas the communication costs may be reduced on the costs of the solution performance.

Furthermore, we were interested in two more measures in order to interpret the experimental results more thoroughly.

- **Average travel costs (avgTC):** This measure represents the average costs for the movement of one AGV participating in the solution, i. e. the average travel costs per AGV that has at least once left the depot. In certain scenarios this measure provides more detailed insight into the solution.
- **Average workload (avgWL):** This measure represents the average workload of one AGV participating in the solution, i. e. the average number of goods transported per AGV that has at least transported one good during the solution. Similarly, in certain scenarios this measure provides more detailed insight into the solution.

Another interesting measure is the minimal amount of vehicles required for the solution to a PDP scenario, as the purchase costs for a single AGV are extremely high. However, this number cannot be directly measured in the experiment settings. The above measures, however, give an indication, which experiment series require the less number of vehicles for the solution.

Table 8.4 lists the most important parameters and values that were used for the experiment execution. Obviously, there are quite a number of parameters that can be changed in PIC. Depending on the respective environment of the PDP scenarios these parameters have been set manually in a kind of best practice, based on the experiences we have made with PIC. However, subtle changes in these parameters also change the solutions significantly. Unfortunately, the non-linearity in the results of self-organizing emergent solutions prevents the application of adequate optimization approaches such as evolutionary algorithms. The optimality of the parameters depend significantly on the number and the location of the tasks, the number of vehicles, and the size of the environment. Optimizing these parameters thus is left for future work. Please note, in the $SAPg$ series all infochemicals have been configured similar to the configuration of synomones in the respective environment, while

the emission rate of all agents has been configured similar to the configuration of pollinator agents in the respective environment.

Name	Description	Environment				
		TW-Env 1	TW-Env 2	TW-Env 3	TW-Env 4	AM-Env
$conc$	infochemical emission concentration	35	45	55	75	60
rc	reward concentration of flowers	10	10	10	10	10
id	idle time of pollinator agents	1	1	1	1	1
λ	utility adjustment factor	10^6	10^6	10^6	10^6	10^6
Emission rates						
emr_{pto}	emission rate of pollinator agents	5	5	5	5	5
emr_{pnz}	emission rate of pollenizer agents	10	10	10	10	10
emr_{hv}	emission rate of hive agents	100	100	100	100	100
Pheromones						
ef	evaporation factor	0.30	0.30	0.26	0.30	0.20
$coef$	diffusion coefficient	3.00	3.00	3.50	3.00	4.00
$conc_{thresh}$	threshold concentration	6.00	5.00	5.00	5.00	5.00
Allomones						
ef	evaporation factor	0.35	0.35	0.35	0.45	0.75
$coef$	diffusion coefficient	2.50	2.00	2.00	2.00	1.00
$conc_{thresh}$	threshold concentration	0.50	6.00	6.00	1.00	0.15
Kairomones						
ef	evaporation factor	0.96	0.96	0.96	0.95	0.98
$coef$	diffusion coefficient	1.00	1.00	1.00	1.00	1.00
$conc_{thresh}$	threshold concentration	0.30	0.30	0.30	0.40	0.50
Synomones						
ef	evaporation factor	0.71	0.71	0.71	0.68	0.75
$coef$	diffusion coefficient	1.00	1.00	1.00	1.00	1.00
$conc_{thresh}$	threshold concentration	0.50	0.50	0.50	0.10	0.15

Table 8.4: Parameter values for the evaluation of the PIC mechanism

Due to these values, pheromones and allomones are configured to have a short lifetime only, synomones have a medium lifetime, whereas kairomones have a long lifetime. The propagation range of pheromones is configured to be small, allomones have a medium propagation range, whereas synomones and kairomones have a wide propagation range (except for the SAPg series). To overcome statistical anomalies and to account for the randomness used in the scenarios, we have executed every scenario 100 times and averaged the results at the end.

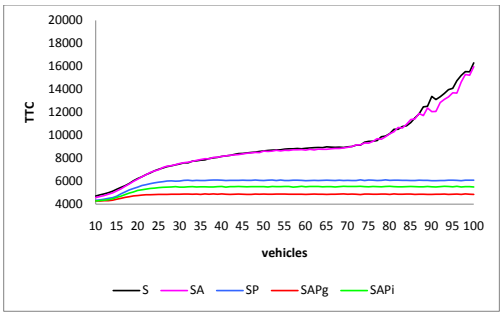
8.3.3 Experimental Results

Figures 8.10 – 8.21 depict the experimental results for each of the PDP scenarios of the first case study (see Table 8.2), starting with 100 transportation requests on the *TW-Env 1* environment up to 500 transportation requests on the *TW-Env 4* environment. Figures 8.22 – 8.27 depict the experimental results for each of the PDP scenarios of the second case study (see Table 8.3), starting from 100 up to 500 transportation requests on the *AM-Env* environment. In all figures, the black line represents the results of the experiment series *S*, the purple line the results of the *SA* series, the blue line the results of the *SP* series, the red line the results of the *SAPg* series, and the green line the results of the *SAPi* series. Due to the huge size and high dynamics of our PDP scenarios, there exist no suitable solution approaches able to determine the optimal solution, e. g. in terms of TTC or CT, nor the optimal number of vehicles required for the solution (cf. Subsection 7.1.3). Consequently, appropriate benchmark instances and results do not exist as well. However, the black lined *S* series series can be considered as a kind of benchmark for the state of the art, as this series uses only one type of infochemical, whereas all other series make extensions to this series based on the IBC approach.

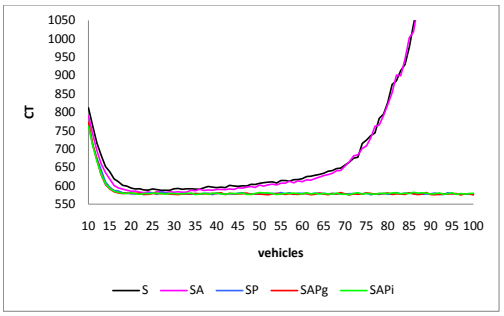
8.3.3.1 Environment *TW-Env 1* with 100 Transportation Requests

Figure 8.10 depicts the experimental results for the scenario *tw-env1-100* having 100 transportation requests (tasks) on the smallest environment *TW-Env 1*. The total travel costs (TTC) of this PDP scenario are shown in Figure 8.10(a). Whereas the TTC permanently increase for the *S* and *SA* series, they remain nearly constant for the *SP*, *SAPg*, and *SAPi* series starting with 20 vehicles and higher. These results clearly demonstrate that due to the use of multiple types of infochemicals for this specific number of tasks in this specific environment, the TTC can be significantly reduced. Apparently, in particular the use of pheromones (in *SP*/*SAPg*/*SAPi*) for the coordination between vehicles improves the overall solution significantly. By contrast, the use of allomones in *SA* improves the solution only marginally compared to the use of synomones only (in *S*). The reason for the marginal improvement is that due to the missing coordination between the vehicles too many vehicles are present in this small environment, so that the effect of allomones very often fizzles out. Due to the random generation of the transportation requests and the random choosing of locations, the solution in a few instances is even more worse. However, the use of allomones in *SAPg*/*SAPi* compared to the solution of synomones and pheromones only (in *SP*) demonstrates a clear improvement. Because in these two series already a coordination between the vehicles takes places, allomones additionally improve the coordination. Apparently, as more frequent the information exchange between the vehicles (in *SAPg*), as better the result with regard to TTC.

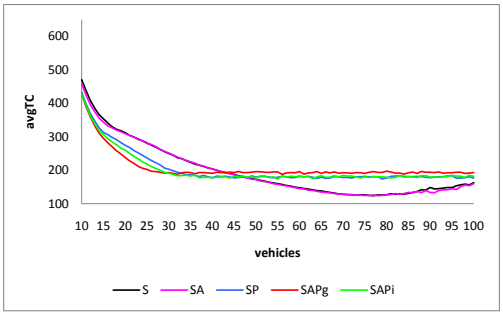
A similar result is shown by Figure 8.10(b) depicting the completion time (CT) of this PDP scenario. Whereas a minimum CT is achieved by around 20 vehicles for all experiment series, the CT in the *S* and *SA* series from then on start to



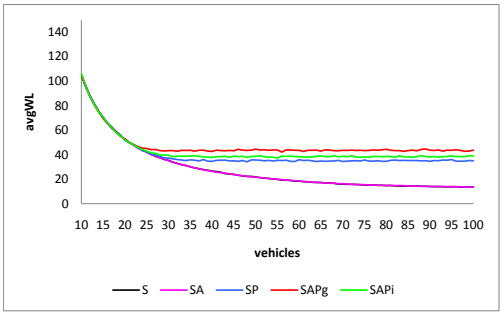
(a) Total travel costs



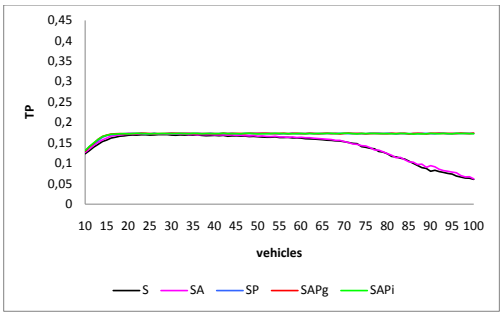
(b) Completion time



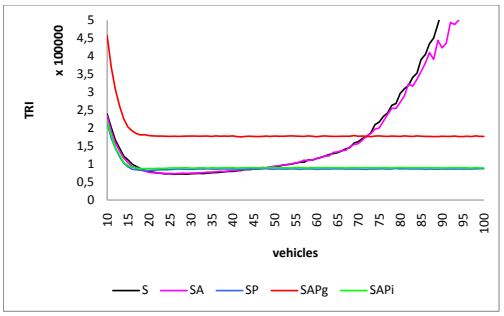
(c) Average travel costs per vehicle



(d) Average workload per vehicle



(e) Throughput



(f) Total rate of infochemicals

Figure 8.10: Experimental results for 100 transportation requests in *TW-Env 1*

increase moderately, from 70 vehicles on even quite strongly, whereas the results of the other series remain nearly constant again. This gives another indication that the coordination between the vehicles in the S and SA series is not so good, so that too many vehicles are present in the environment (which results from the reactiveness of the agents as a runtime insufficiency). On the other side, the results indicate that for this specific parameter configuration this PDP scenario cannot be solved faster by the current specification of the PIC mechanism.

Figure 8.10(c) depicts the average travel costs (avgTC) of a single vehicle. In this figure, the avgTC in the *S* and *SA* series are lower compared to the other series starting with 50 vehicles and higher. At a first glance, this result seems to worsen the overall solutions when more types of infochemicals are used. On a closer inspection, however, this figure proves the indication provided by the first two measures: due to the better coordination in the *SP*, *SAPg*, and *SAPi* series, vehicles that are not required for the solution to a PDP are able to remain in the depot, which in the end generates less TTC of all vehicles but as a consequence higher avgTC for a single vehicle participating in the solution. Moreover, these unused vehicles do not unnecessarily congest the environment, which in particular in such small environments as *TW-Env 1* is the reason for the higher CT of the *S* and *SA* series at the end. Consequently, the avgTC of the *SAPg* series are the highest starting with 44 vehicles and higher but remain constant, as the number of vehicles required for the solution does not change. The results of the avgTC measure between 10 and 44 vehicles have to be interpreted the other way round. The better the coordination between the vehicles respectively the pickup/delivery stations and the vehicles, the less unnecessary movements are generated, which reduce the avgTC of a single vehicle.

Figure 8.10(d) depicts the results of the average workload (avgWL) of a single vehicle. Again, the better the coordination between the agents, the less vehicles are being used respectively leave the depot unnecessarily, the higher the avgWL of the vehicles participating in the solution. This becomes even more apparent by the results of the throughput (TP) measure depicted in Figure 8.10(e). Whereas in particular up to the point of 20 vehicles the results are nearly the same for all series, the throughput results for the *S* and *SA* series begin to worsen again as the number of available vehicles increases. This is another indication that the environment in these series becomes congested by superfluous vehicles.

Figure 8.10(f) depicts the total rate of infochemicals (TRI) generated during the solution, in other words the communication costs to produce the solution. Apparently, the costs for producing the solution by the *SAPg* series is considerably higher than the costs for producing the solution of the *SP* and *SAPi* series. This is a result of the high frequency of information exchange. The costs of the *S* and *SA* series are even for a short period the lowest but towards the end increase steadily and quite strongly again.

8.3.3.2 Environment *TW-Env 1* with 250 Transportation Requests

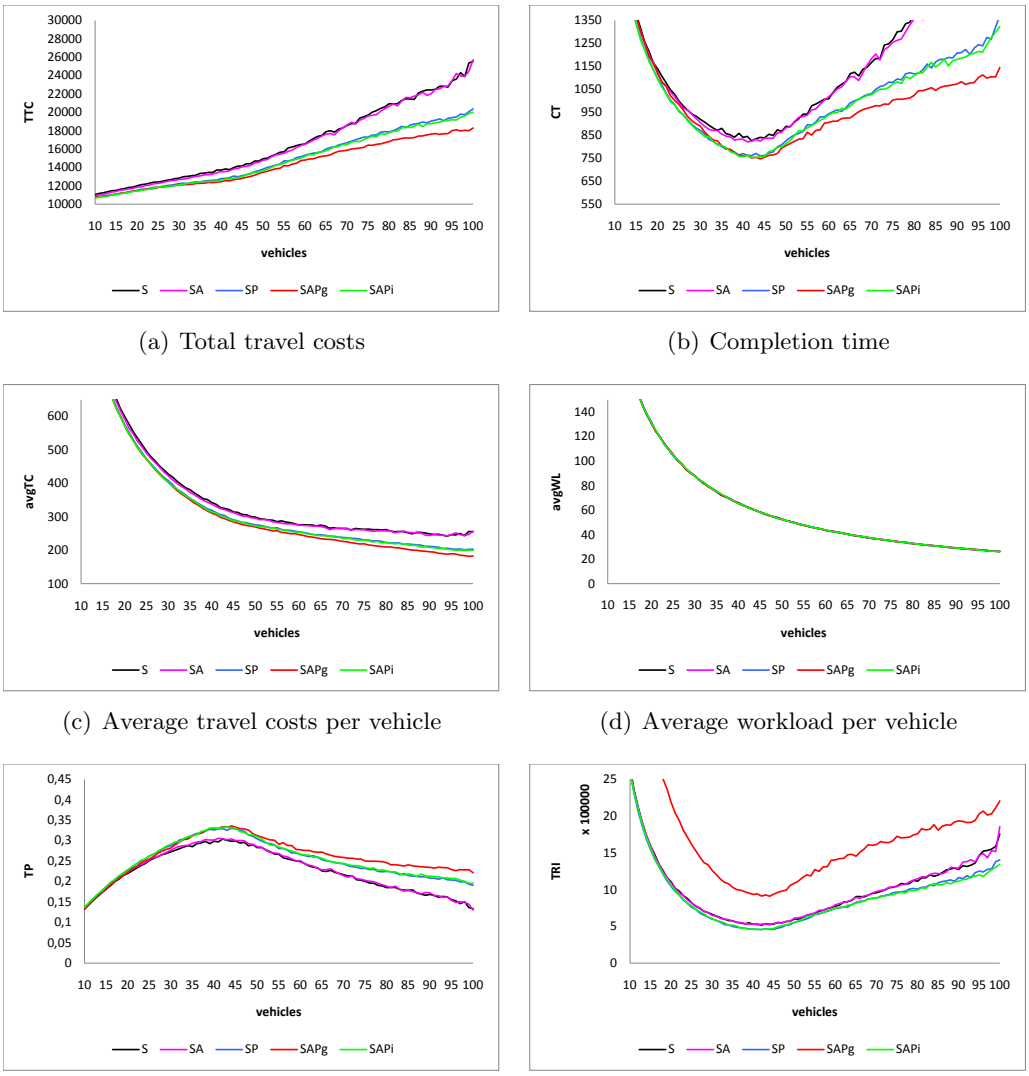
With regard to the results depicted in Figure 8.11 for the scenario *tw-env1-250*, it becomes apparent that the coordination by more types of infochemicals is again able to produce better results, even for 250 transportation requests. The CT (Figure 8.11(b)), the TP (Figure 8.11(e)), and the TRI (Figure 8.11(f)) are at their minima respectively maxima around the value of 42, which indicates that this is a preferable number of vehicles for the solution. As soon as more vehicles are available, the coordination however is not sufficient enough to prevent unnecessary movements by the vehicles and an overcrowding of pathways, indicated by the increasing CT and TRI as well as the decreasing TP. Also, around the value of 42, the slopes of the TTC lines (Figure 8.11(a)) of the *SP*, *SAPi*, and *SAPg* series are lower. The avgTC (Figure 8.11(c)) steadily decrease for all series, but have a buckling in their negative slope around this value as well. The avgWL results (Figure 8.11(d)) in contrast are nearly similar for all series. Due the lower TTC in particular of the *SP*, *SAPi*, and *SAPg* series, however, the avgWL results indicate that these series require less vehicles for the solution as in the *S* and *SA* series. The effect of using pheromones again is significant in all results, whereas the effect of the additional use of allomones is only marginal in all results.

8.3.3.3 Environment *TW-Env 1* with 500 Transportation Requests

The results for 500 transportation requests in the scenario *tw-env1-500* (depicted in Figure 8.12) have the same trends as for 250 transportation requests. Again, using different types of infochemicals results in better TTC, CT, and TP, but the coordination again is not sufficient enough for avoiding unnecessary movements by the vehicles as soon as more than 42 vehicles are available. When comparing the absolute results with the results for 100 respectively 250 transportation requests, it becomes apparent that the TTC nearly increase by the same factor as the transportation requests. Similarly, the CT increases as well, however, by a lower factor. The TP results indicate that a higher throughput than 0,31-0,32 goods per iteration is not to achieve in this small environment due to the bottleneck around the depot. For 100 transportation requests, however, this throughput can not be achieved due to the lower number of tasks to fulfill. The congestion of the environment is again documented by the TRI, as they increase very strongly with an increasing amount of transportation requests.

8.3.3.4 Environment *TW-Env 2* with 100 Transportation Requests

Figure 8.13 depicts the experimental results for the scenario *tw-env2-100* having 100 transportation requests (tasks) in the environment *TW-Env 2*. With regard to the TTC, CT, avgTC, avgWL, and TP the absolute results are quite similar to the results for the fulfillment of 100 transportation requests in the smaller environment *TW-Env 1* (see Figure 8.10). Only the TRI is higher for this environment, mainly due to the higher amount of locations in the environment. Because in this



(a) Total travel costs

(b) Completion time

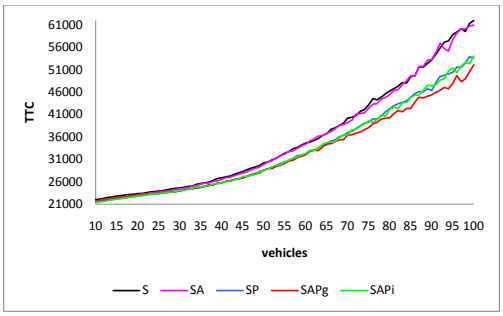
(c) Average travel costs per vehicle

(d) Average workload per vehicle

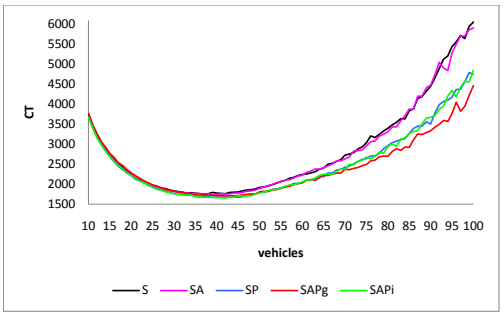
(e) Throughput

(f) Total rate of infochemicals

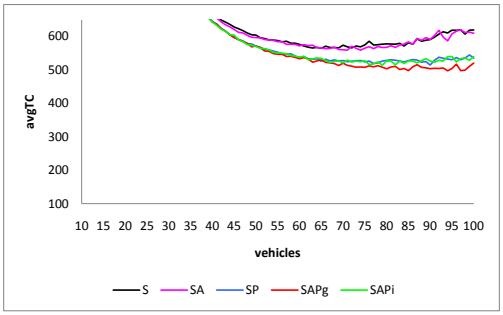
Figure 8.11: Experimental results for 250 transportation requests in *TW-Env 1*



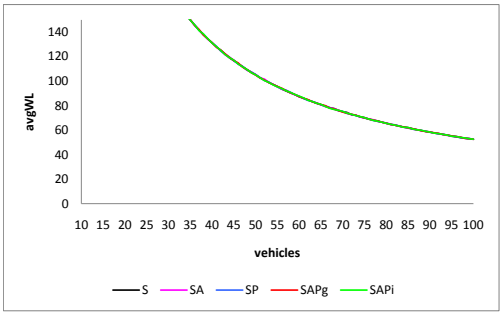
(a) Total travel costs



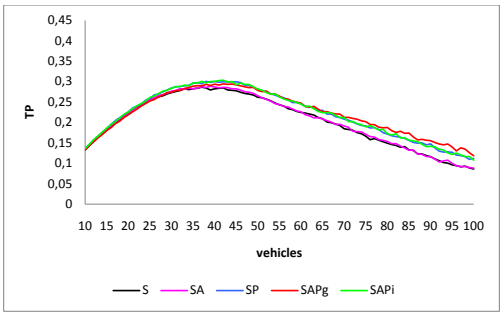
(b) Completion time



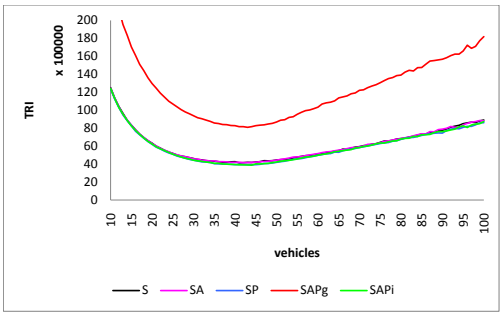
(c) Average travel costs per vehicle



(d) Average workload per vehicle



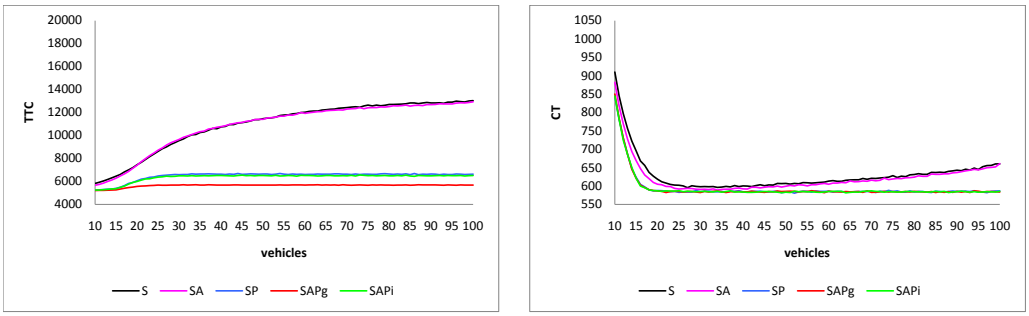
(e) Throughput



(f) Total rate of infochemicals

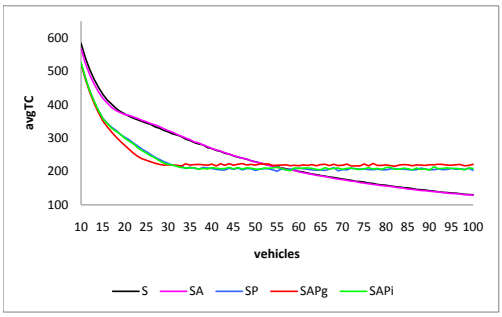
Figure 8.12: Experimental results for 500 transportation requests in *TW-Env 1*

environment the vehicles have more space available, in particular there are more locations available than vehicles participating, congestions in the environment only arise around the depot, but do not have such significant effects as in the smaller environment. Thus, for the *S* and *SA* series the TTC, CT, and TRI do not exhibit such a strong increase with an increasing amount of vehicles any more, while the TP of these series do not decrease so strongly. However, all of these measures are not able to remain constant again, as it is the case for the *SP*, *SAPg*, and *SAPi* series.

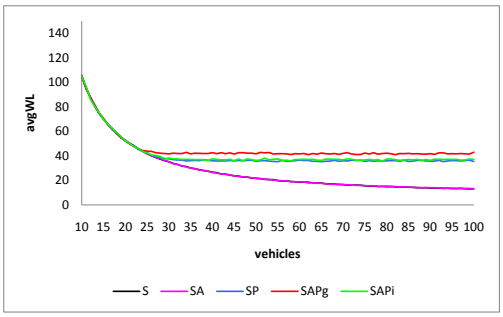


(a) Total travel costs

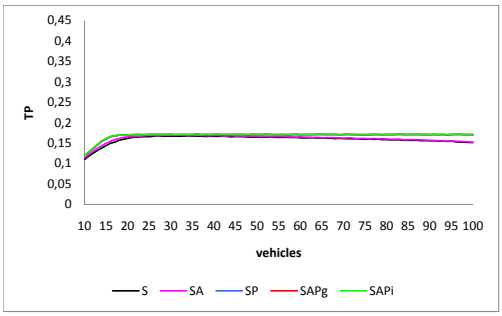
(b) Completion time



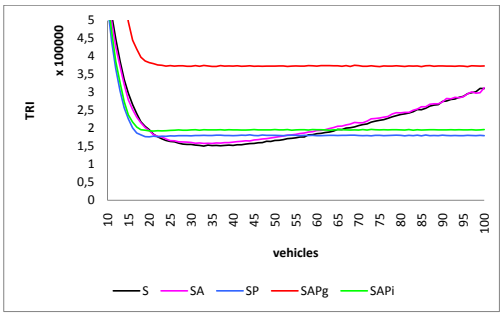
(c) Average travel costs per vehicle



(d) Average workload per vehicle



(e) Throughput



(f) Total rate of infochemicals

Figure 8.13: Experimental results for 100 transportation requests in *TW-Env 2*

8.3.3.5 Environment *TW-Env 2* with 250 Transportation Requests

Figure 8.14 depicts the results for the scenario *tw-env2-250* having 250 transportation requests on the environment *TW-Env 2*. Compared to the results of the smaller environment (see Figure 8.11), again several similarities in the trends can be observed. Although the environment has twice as many locations as the smaller environment, which forces the vehicles to travel slightly longer distances (see TTC in Figure 8.14(a)), the minima of the CT of all experiment series are lower (see Figure 8.14(b)) respectively the maxima of the TP are higher (see Figure 8.14(e)) compared to the corresponding results in the smaller environment. This again indicates that the vehicles have more space available, which reduces congestions and improves the solutions in all experiment series. The most obvious difference between these results is however, that from a specific number of vehicles on (around 45) only the *SAPg* series is able to achieve nearly constant results for all measures, which indicates that the configurations of the other series are not good enough to prevent from inefficiencies in this bigger environment.

8.3.3.6 Environment *TW-Env 2* with 500 Transportation Requests

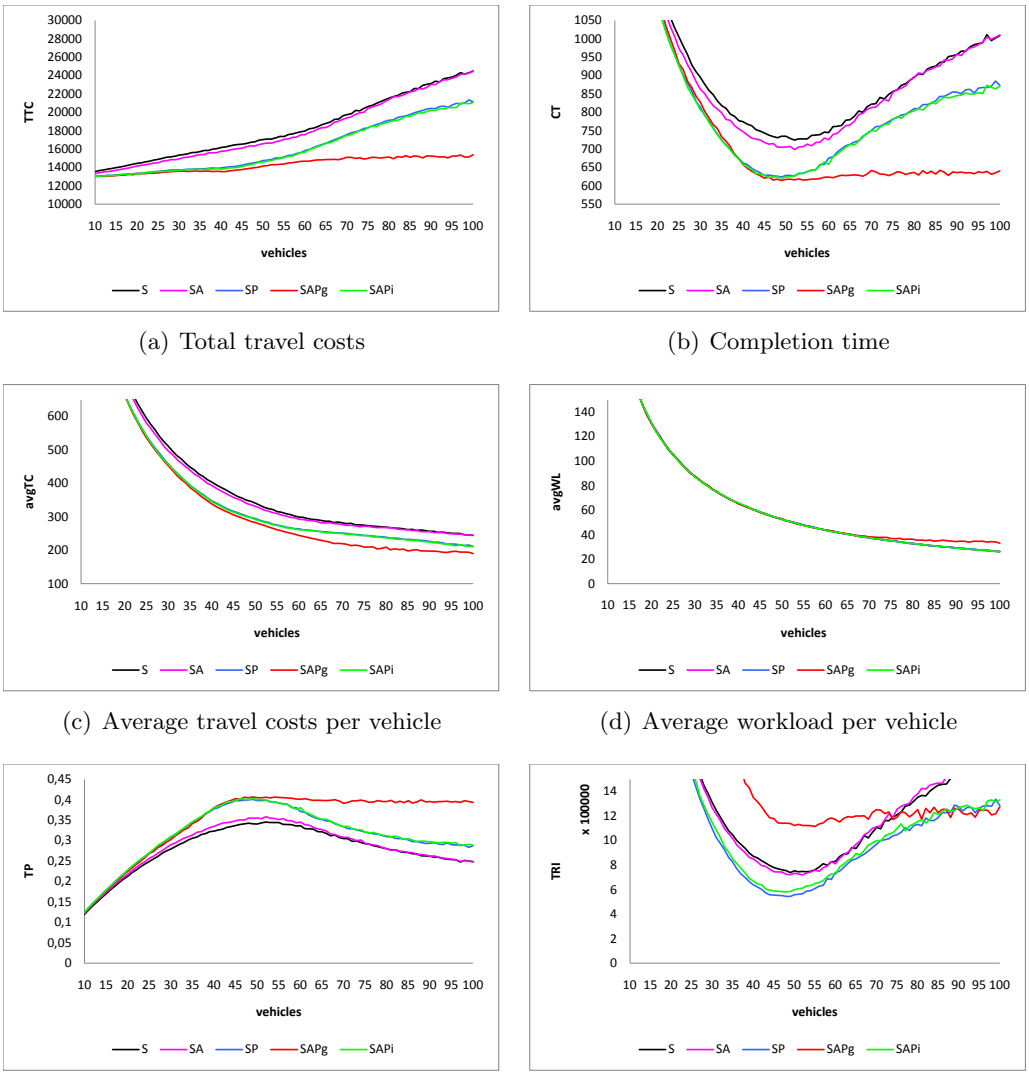
Figure 8.15 depicts the results for the scenario *tw-env2-500* having 500 transportation requests on the environment *TW-Env 2*. As the number of transportation requests increase in this second environment, even the *SAPg* series is not able to produce constant results any more, although again the coordination by means of different typed infochemicals achieves better results than the coordination by only one type of infochemical. Compared to the results of the smaller environment (see Figure 8.12), the scenarios again are solved faster by all series (see CT), while generating a higher TP. The trends of the resulting graphs, however, now allow an easier conclusion on the minimal amount of vehicles required for the solution of this scenario (around 50).

8.3.3.7 Environment *TW-Env 3* with 100 Transportation Requests

Figure 8.16 depicts the results for the scenario *tw-env3-100* having 100 transportation requests on the environment *TW-Env 3*. Again, most of the experiment series are able to achieve nearly constant results, starting around 20 vehicles and higher. Even though the environment has about twice as many locations as the environment *TW-Env 2* and four times as many locations as the environment *TW-Env 1* (which is the reason why the TTC and the TRI are higher compared to the TTC and TRI in those environments), the avgTC and avgWL as well as in particular the CT and TP are nearly the same as in the smaller environments.

8.3.3.8 Environment *TW-Env 3* with 250 Transportation Requests

Figure 8.17 depicts the results for the scenario *tw-env3-250* having 250 transportation requests on the environment *TW-Env 3*. It becomes obvious, that the larger



(a) Total travel costs

(b) Completion time

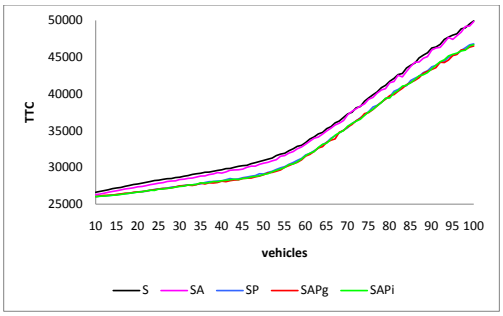
(c) Average travel costs per vehicle

(d) Average workload per vehicle

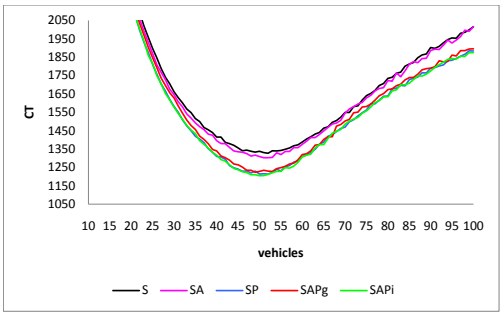
(e) Throughput

(f) Total rate of infochemicals

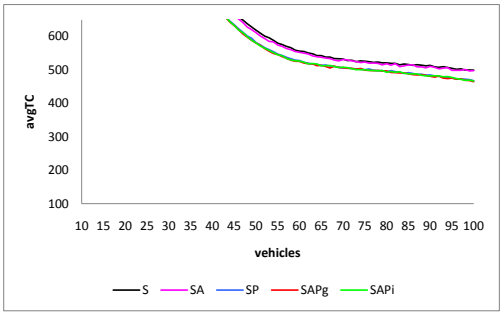
Figure 8.14: Experimental results for 250 transportation requests in *TW-Env 2*



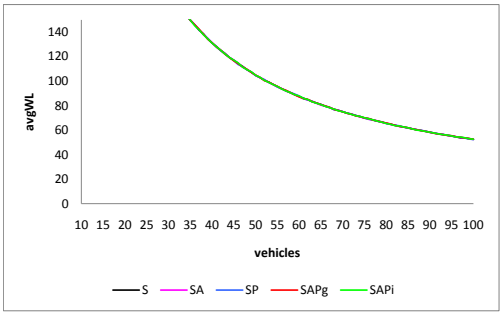
(a) Total travel costs



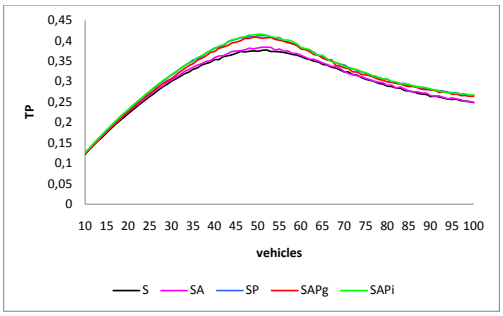
(b) Completion time



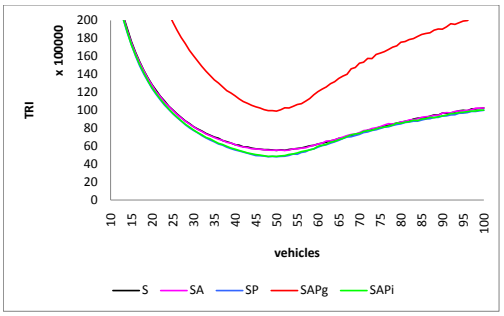
(c) Average travel costs per vehicle



(d) Average workload per vehicle

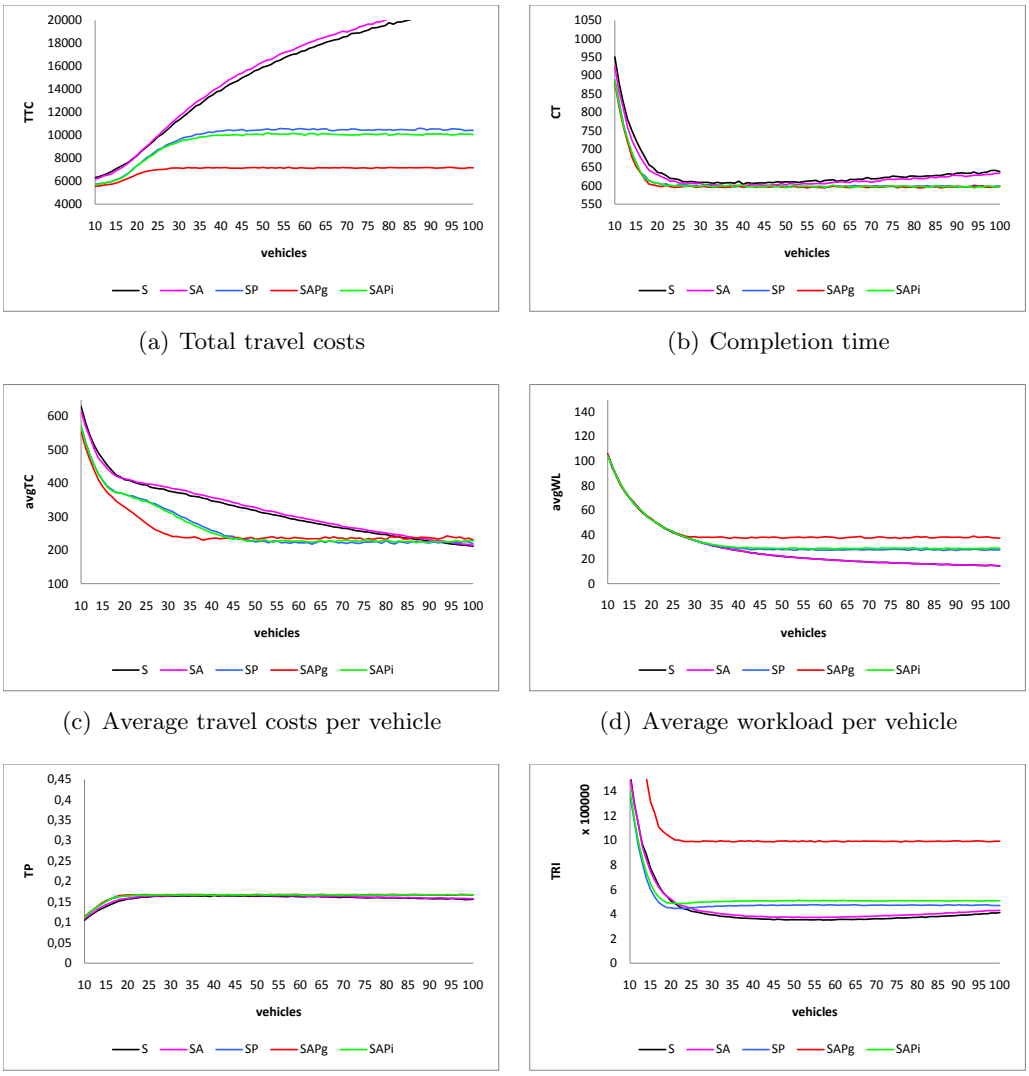


(e) Throughput



(f) Total rate of infochemicals

Figure 8.15: Experimental results for 500 transportation requests in *TW-Env 2*



(a) Total travel costs

(b) Completion time

(c) Average travel costs per vehicle

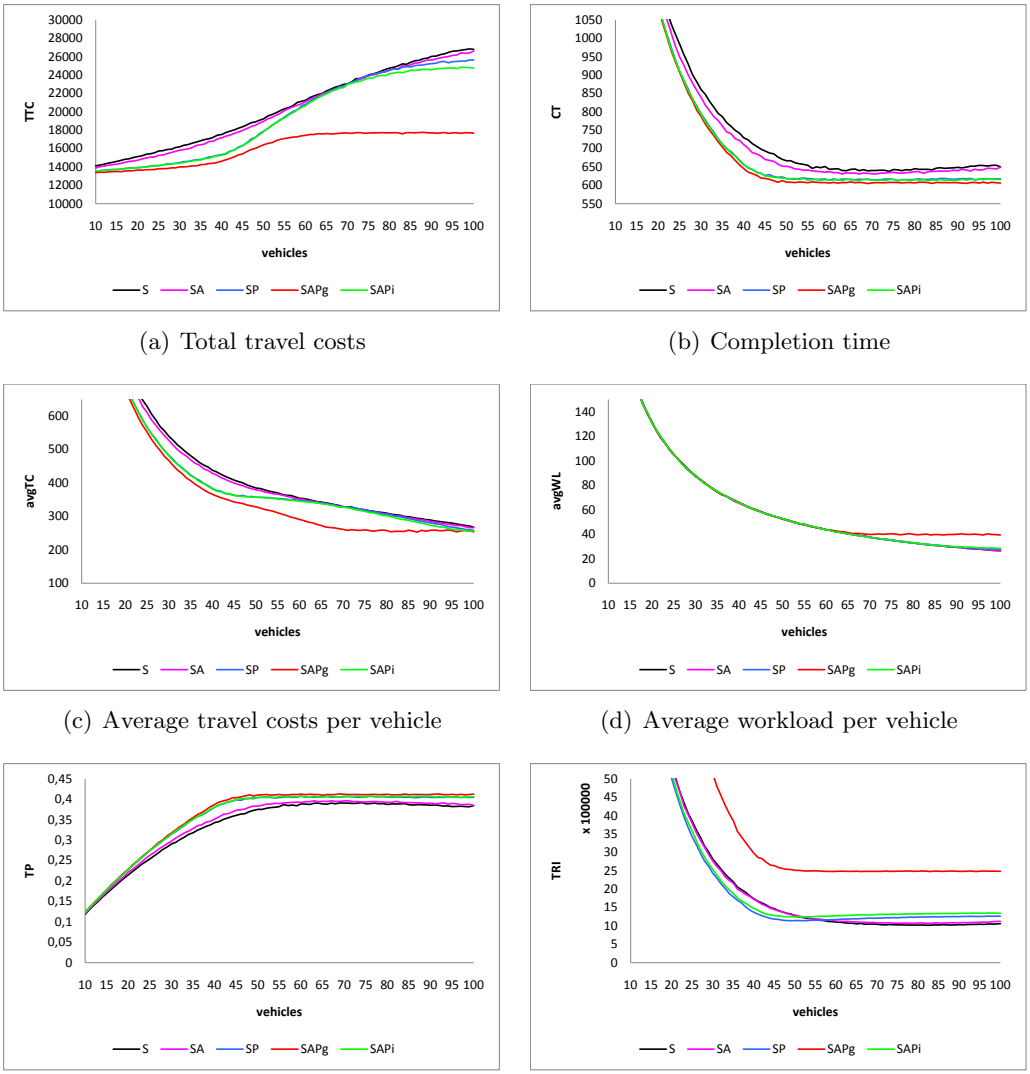
(d) Average workload per vehicle

(e) Throughput

(f) Total rate of infochemicals

Figure 8.16: Experimental results for 100 transportation requests in *TW-Env 3*

the size of the environment, the more experiment series are able to produce nearly constant results. Whereas in the environment *TW-Env 1* none of the experiment series were able to do so, in *TW-Env 2* only the SAPg series was able to do so. However, in *TW-Env 3* also the other series finally reach a constant level in almost all measures. An exception is however made by the TTC, where all but the *SAPg* series reach this constant level very late. Furthermore, these levels are quite high compared to the level of the *SAPg* series. In particular the CT for all series is better than in the environment *TW-Env 2*, despite the large size of *TW-Env 3*.



(a) Total travel costs

(b) Completion time

(c) Average travel costs per vehicle

(d) Average workload per vehicle

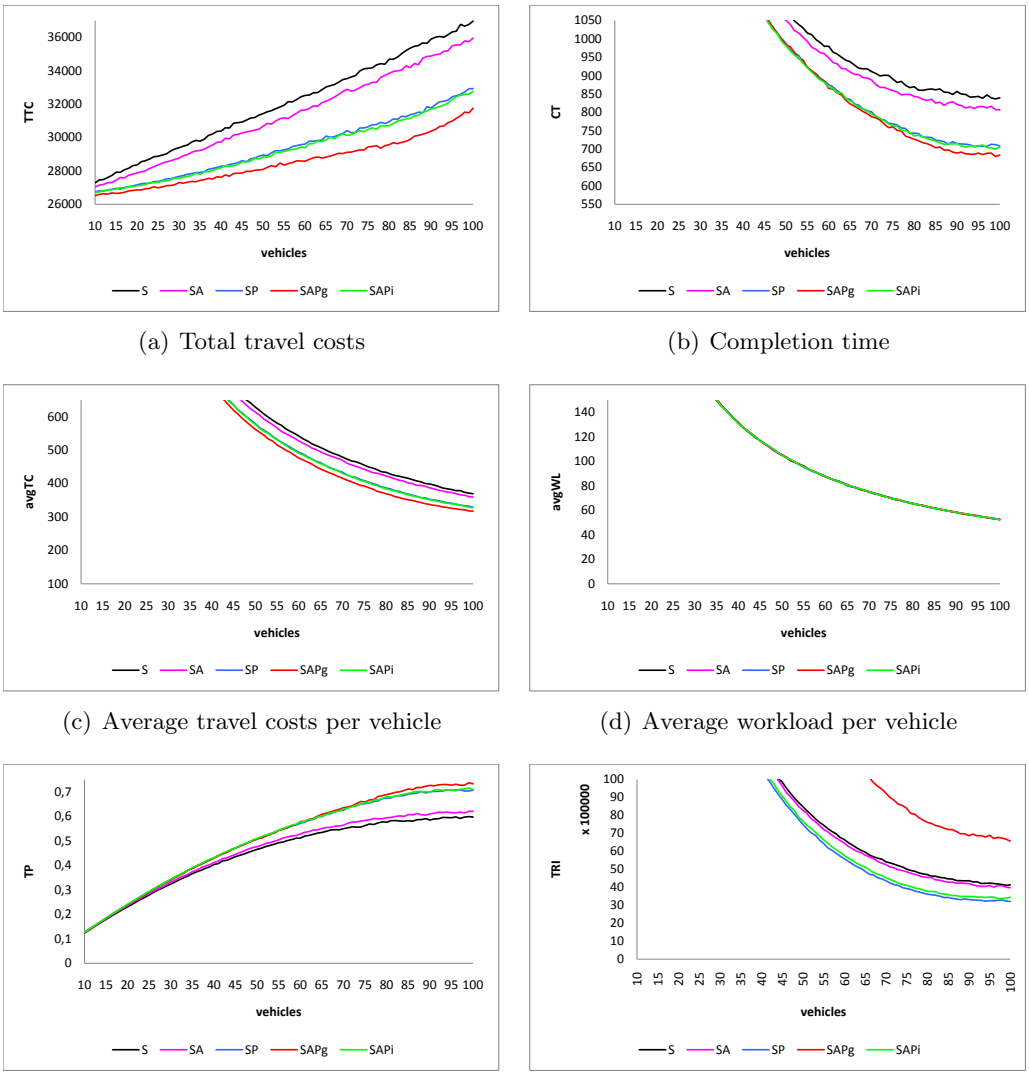
(e) Throughput

(f) Total rate of infochemicals

Figure 8.17: Experimental results for 250 transportation requests in *TW-Env 3*

8.3.3.9 Environment *TW-Env 3* with 500 Transportation Requests

Figure 8.18 depicts the results for the scenario *tw-env3-500* having 500 transportation requests in the environment *TW-Env 3*. When considering Figure 8.18(a) it becomes obvious for the first time that the increase of TTC in all series is nearly linear with an increasing amount of vehicles. Furthermore, because no real minima, maxima, or constant levels appear within the range of 10 to 100 vehicles, this indicates that on the one side no significant congestions arise and on the other side every additional vehicle improves the solution. Of all solutions the *SAPg* series again achieves the best results except for the TRI.



(a) Total travel costs

(b) Completion time

(c) Average travel costs per vehicle

(d) Average workload per vehicle

(e) Throughput

(f) Total rate of infochemicals

Figure 8.18: Experimental results for 500 transportation requests in *TW-Env 3*

8.3.3.10 Environment *TW-Env 4* with 100 Transportation Requests

Figure 8.19 depicts the results for the scenario *tw-env4-100* having 100 transportation requests on the environment *TW-Env 4*. Surprisingly, although this environment has about nine times (890%) the size of the environment *TW-Env 1*, the TP for 100 transportation requests reduces only by 5% on average for all series. Similarly, the CT e. g. for the *SAPg* series only increases by 11% on average compared to the smallest environment, whereas the TTC only increase by 93%. However, the TRI for the *SAPi* series, for instance, increases by the factor 16. Although this value is considerable higher, it could possibly be reduced by an optimzation of the parameters of the infochemicals.

8.3.3.11 Environment *TW-Env 4* with 250 Transportation Requests

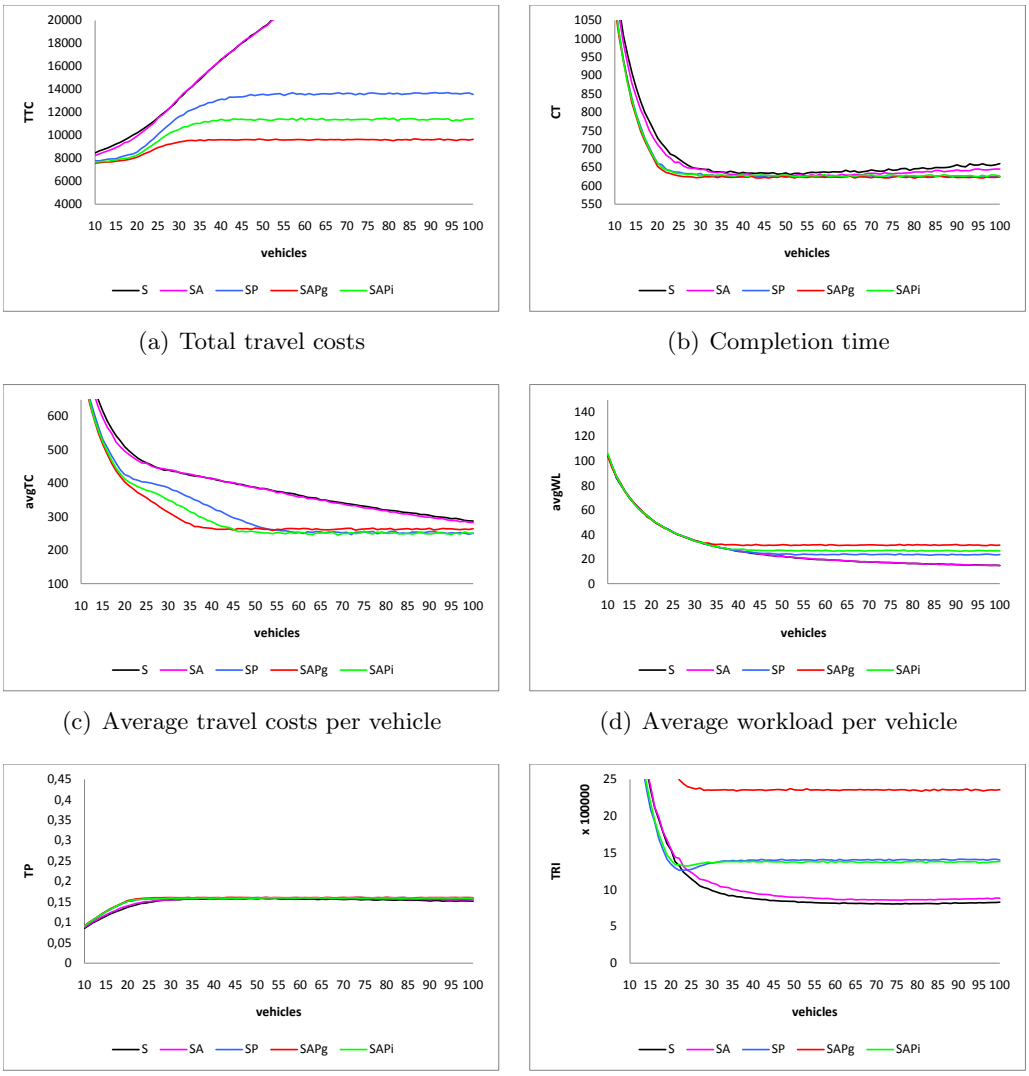
Figure 8.20 depicts the results for the scenario *tw-env4-250* having 250 transportation requests on the environment *TW-Env 4*. The results again indicate that congestions are reduced compared to the environment *TW-Env 1*, which is the reason for the increase of the TP compared to the TP in the environment *TW-Env 1* (see Figure 8.20(e)). However, as the TTC of all experiment series do not increase linearly and the TRI of the *SP* and *SAPi* series have a change in the slope, unnecessary movements of superfluous vehicles occur, starting from 50 vehicles and higher.

8.3.3.12 Environment *TW-Env 4* with 500 Transportation Requests

Figure 8.21 depicts the results for the scenario *tw-env4-500* having 500 transportation requests on the environment *TW-Env 4*. Similar to the results on the smaller environment *TW-Env 3*, the TTC increase linearly with an increasing amount of vehicles, whereas the TP increases more slowly. At the same time, the CT, avgTC, avgWL, and TRI decrease with an increasing amount of vehicles. Thus, in this big environment with this big number of tasks to fulfill, every additional vehicle improves the solution (based on 10 to 100 vehicles).

8.3.3.13 Environment *AM-Env* with 100 Transportation Requests

With regard to the second case study, Figure 8.22 depicts the results for the scenario *am-env-100* having 100 transportation requests on the environment *AM-Env*. Concerning the TTC (Figure 8.22(a)), the trends of all experiment series are very similar to the trends of 100 transportation requests in the corresponding environments of the first case study. However, the improvements of each experiment series compared to the S series now becomes very apparent, whereas the differences between the *SAPg* and the *SAPi* series are only very small any more. With regard to the CT (Figure 8.22(b)), however, one significant difference compared to the prior results of the first case study becomes explicit: the *SAPg* series requires more time to complete the scenario than all other series (at least for more than 33 vehicles).



(a) Total travel costs

(b) Completion time

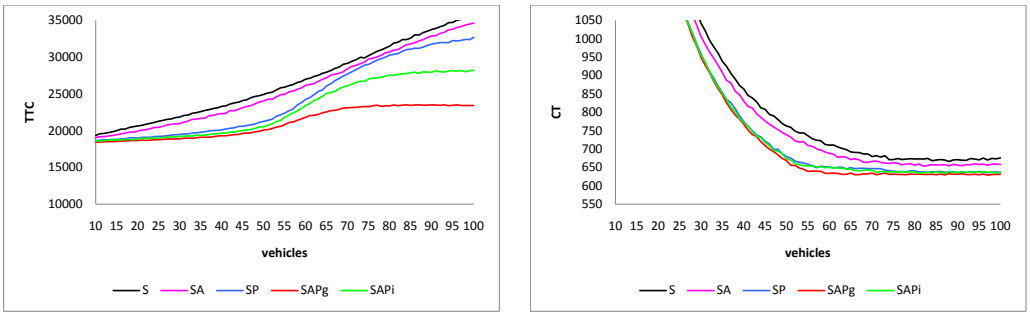
(c) Average travel costs per vehicle

(d) Average workload per vehicle

(e) Throughput

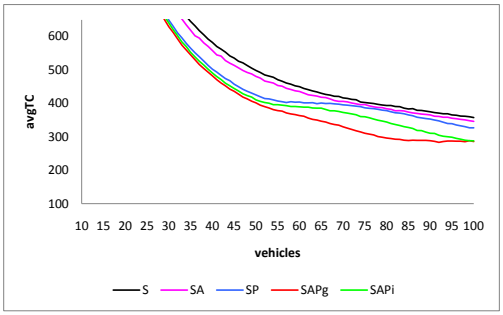
(f) Total rate of infochemicals

Figure 8.19: Experimental results for 100 transportation requests in *TW-Env 4*

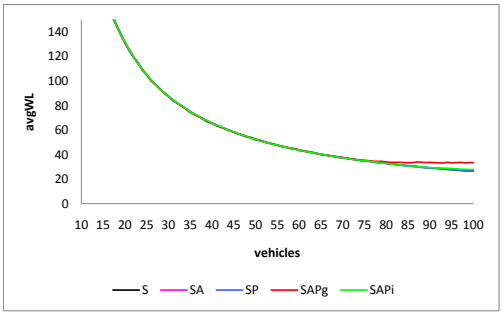


(a) Total travel costs

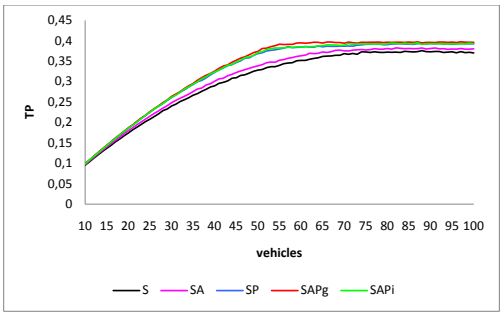
(b) Completion time



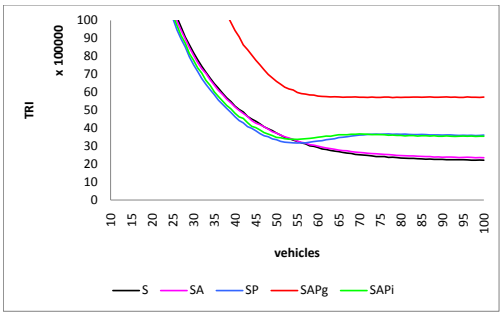
(c) Average travel costs per vehicle



(d) Average workload per vehicle

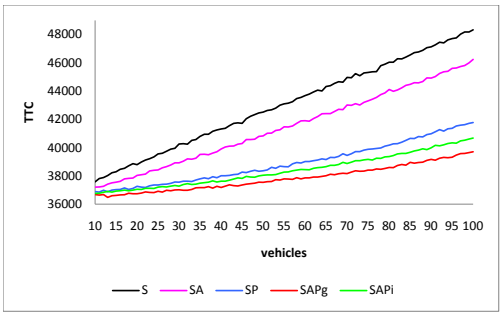


(e) Throughput

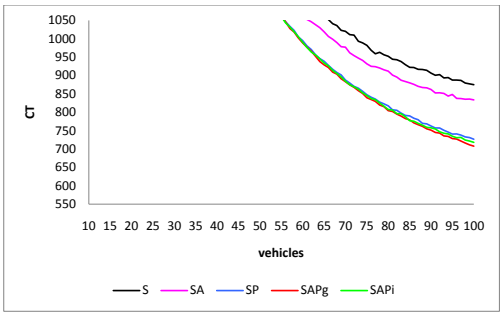


(f) Total rate of infochemicals

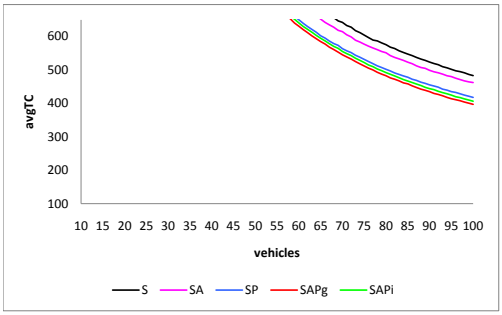
Figure 8.20: Experimental results for 250 transportation requests in *TW-Env 4*



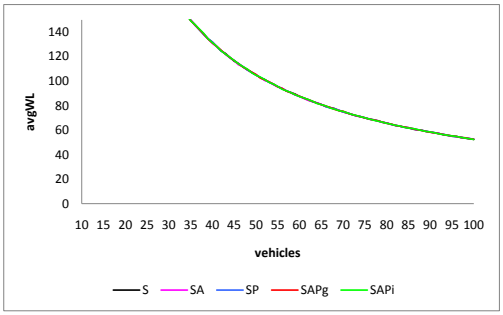
(a) Total travel costs



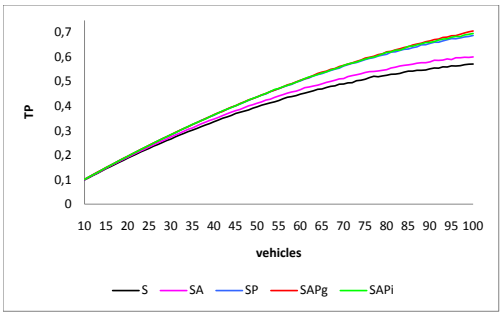
(b) Completion time



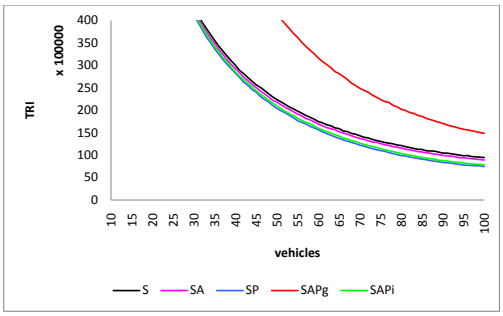
(c) Average travel costs per vehicle



(d) Average workload per vehicle



(e) Throughput



(f) Total rate of infochemicals

Figure 8.21: Experimental results for 500 transportation requests in *TW-Env 4*

Consequently, also the TP of this series is the poorest from that point on. The reason is embedded in the structure of the environment combined with the propagation range and the evaporation factor of pheromones in *SAPg*. Because the pheromones are propagated over a longer distance but evaporate slower than e. g. in the *SAPi* series, pheromones emitted by a vehicle that later switches its targeted pickup location, remain for a longer time in the environment. Because all vehicles have only one to two options to approach a pickup location, they become confused and neglect the respective locations for a long time, which increases the CT. Even though non-evaporating pheromones have confused the vehicles in the first case study as well, there the vehicles had more options to approach a pickup location, which reduced the confusion. The results of the avgTC, avgWL, and TRI measure are rather similar to the prior results, while the series trends have only different intersections.

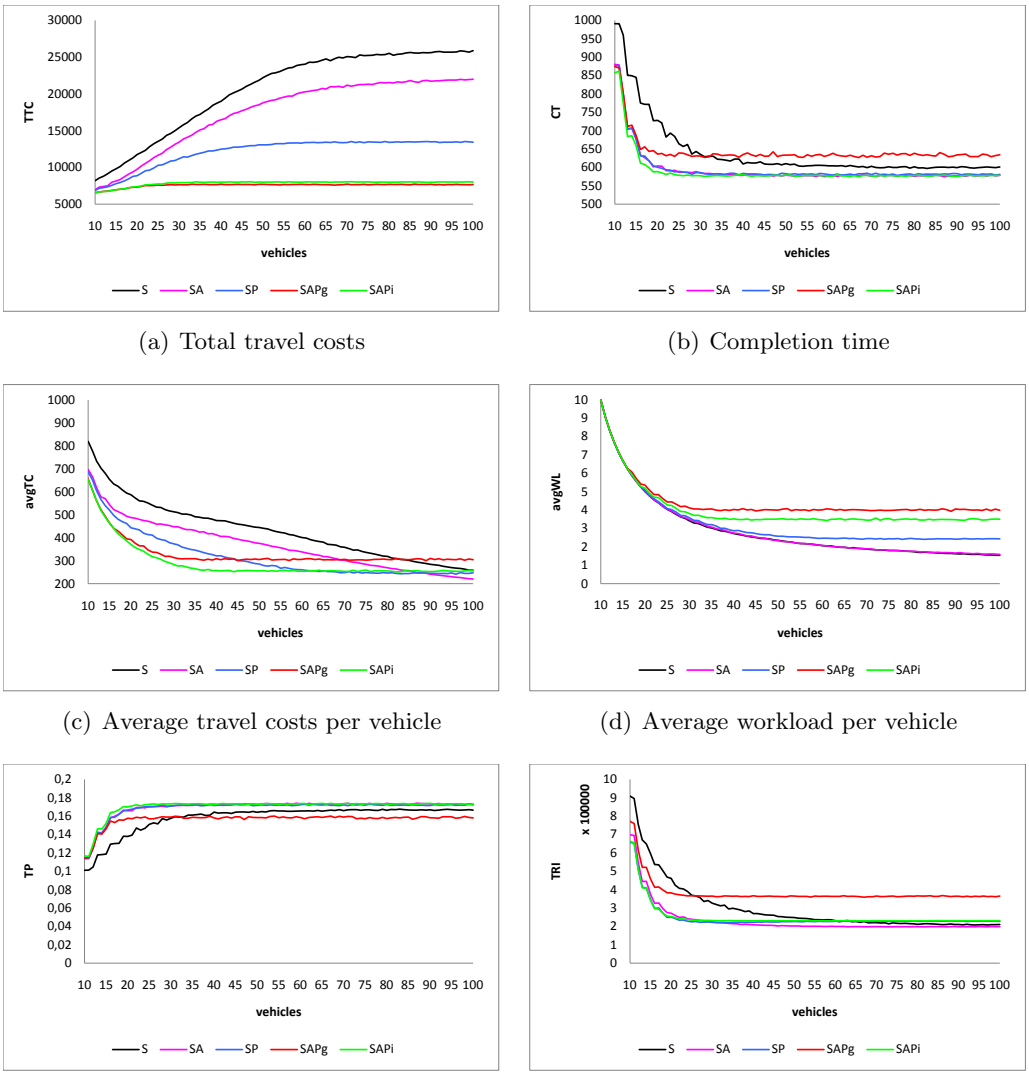
For a better interpretation of the results as well as an illustration of the benefits of IBC, we have additionally measured two values for the second case study:

- **Blocking vehicles (BV):** This measure represents the number of vehicles that leave the depot but do not fulfill any transportation request. In other words, these vehicles block the pathways and yield congestions in the environment.
- **Superfluous vehicles (SV):** This measure represents the number of vehicles that do not even leave the depot. Consequently, they do not block the environment, but are superfluous for the solution.

Figure 8.23(a) and Figure 8.23(b) depict the number of blocking and superfluous vehicles, respectively, for 100 transportation requests on the environment *AM-Env*. Whereas in the *S* and *SA* series the numbers of blocking vehicles increase with an increasing number of available vehicles, the *SP* and in particular the *SAPi* and *SAPg* series were able to limit these numbers to a nearly constant level. The best results have been achieved by the *SAPg* series, where the number of blocking vehicles did not exceed 1. In addition, the SAPg series has the highest number of superfluous vehicles, whereas in the *S* and *SA* series almost all vehicles have fulfilled at least one transportation request, congesting the environment. However, even though the *SAPg* series has the best results in terms of BV and SV – and as a consequence in terms of TTC as well – the CT and TP measures relativize these results. In other words, in particular because the blocking and superfluous vehicles were enabled/-forced to stay in the depot due to the high propagation range and evaporation time of (outdated) pheromones, they did not fulfill any other available transportation requests in this time.

8.3.3.14 Environment *AM-Env* with 250 Transportation Requests

Figure 8.24 depicts the results for the scenario *am-env-250* having 250 transportation requests on the environment *AM-Env*. Although the curves are slightly shifted



(a) Total travel costs

(b) Completion time

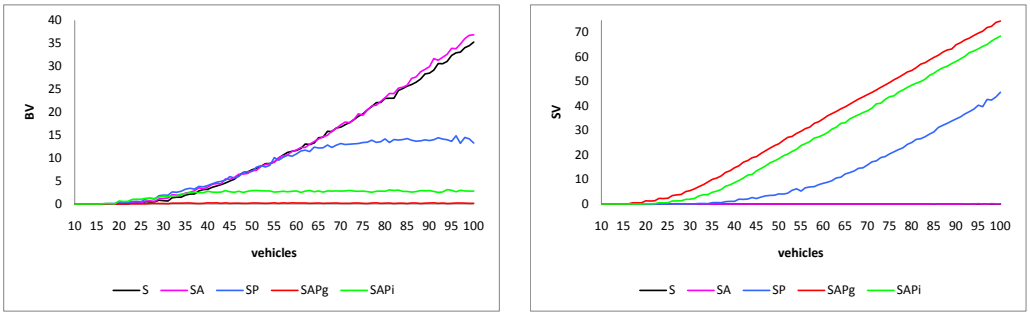
(c) Average travel costs per vehicle

(d) Average workload per vehicle

(e) Throughput

(f) Total rate of infochemicals

Figure 8.22: Experimental results for 100 transportation requests in *AM-Env*



(a) Blocking vehicles (b) Superfluous vehicles

Figure 8.23: Extended results for 100 transportation requests in *AM-Env*

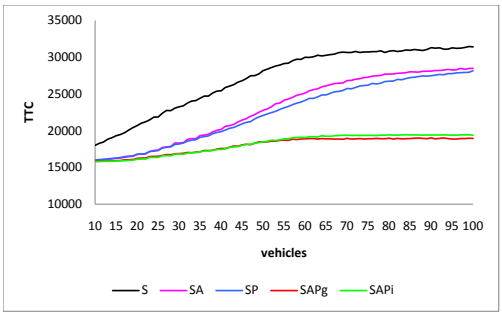
to the upper right, the trends of all series look fairly the same as for 100 transportation requests. However, in contrast to prior results, the curves of the CT and TP measure exhibit at least for a limited range a steplike behavior. This is a result of the distribution policy for agents and transportation requests, i. e. the balancing of vehicles and transportation requests on the available types. Because in these scenarios there have been three types available, the width of a step is 'three vehicles' as well. In other words, from the viewpoint of one group of vehicles of the same type, the number of vehicles of this group increases only after all other groups of vehicles have reached the same size. As a consequence, only then the same number of transportations request could be handled by more vehicles, which results in the increase of TP respectively the decrease of CT.

In contrast to the results of 100 transportation requests, however, the results for the BV (see Figure 8.25(a)) measure look different to some extent. Whereas the number of blocking vehicles is again the lowest for the *SAPg* series, it is now higher for the *SP* series and in particular the *SAPi* series, even higher than for the *S* and *SA* series. Although only up to 4% respectively 9% of the vehicles are blocking, this shows that a decentralized coordination is not able to produce a perfect coordination in all situations (see the runtime insufficiencies in Section 5.1). However, with regard to the good other results (TTC, CT, etc.) of the *SAPi* series, these blocking vehicles carry no special weight.

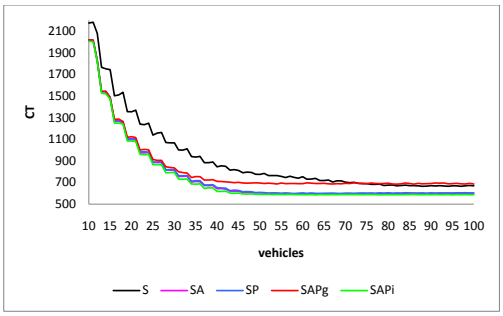
By contrast, the results for the SV (see Figure 8.25(b)) relativize the weakness of the *SAPi* series. Only the *SAPi* and the *SAPg* series were able to fulfill all 250 transportation requests with a reduced amount of vehicles, whereas the insufficient coordination in the *SP*, *SA*, and *S* series required almost all vehicles.

8.3.3.15 Environment *AM-Env* with 500 Transportation Requests

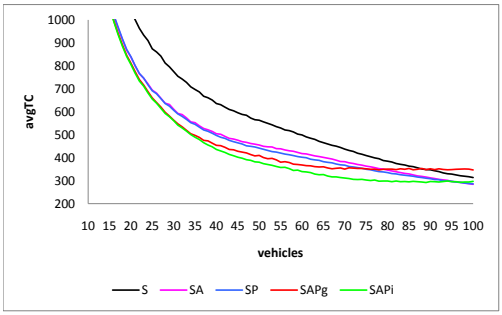
Figure 8.26 depicts the results for the scenario *am-env-500* having 500 transportation requests on the environment *AM-Env*. For all measures, all series are able to improve the solution considerably compared to the benchmarking *S* series, while for this



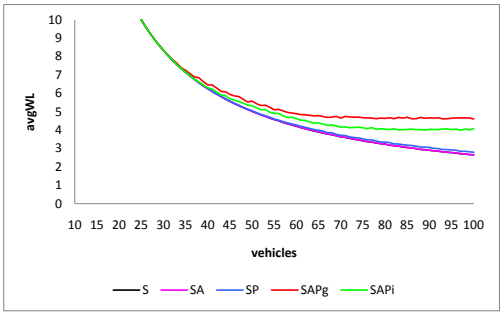
(a) Total travel costs



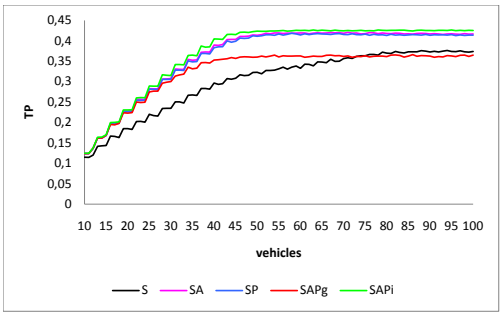
(b) Completion time



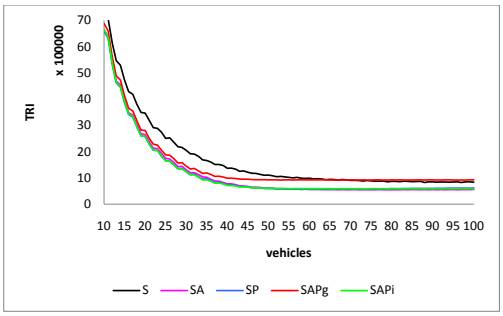
(c) Average travel costs per vehicle



(d) Average workload per vehicle

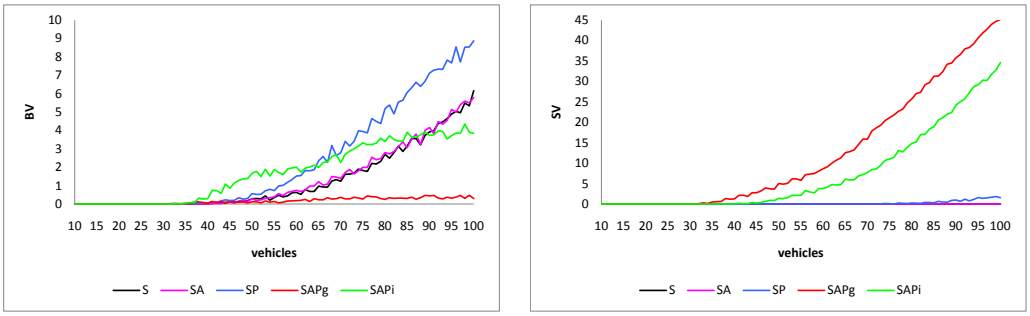


(e) Throughput



(f) Total rate of infochemicals

Figure 8.24: Experimental results for 250 transportation requests in *AM-Env*



(a) Blocking vehicles (b) Superfluous vehicles

Figure 8.25: Extended results for 250 transportation requests in *AM-Env*

number of transportation request for the first time the SAPi series achieves the best results for all measures. In particular in the TP measure (where the steplike behavior now is even more distinctive) the performance differences become apparent.

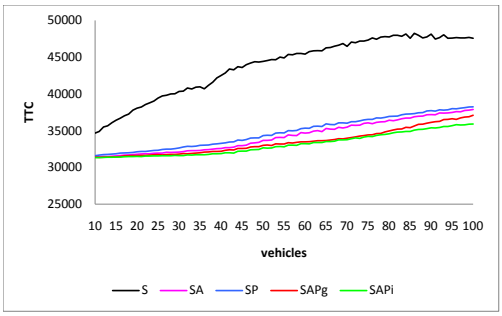
Due to the high number of transportation requests compared to the number of available agents, nearly all agents were required to fulfill the tasks, which results in a negligible number of blocking and superfluous vehicles (see Figure 8.27(a) and Figure 8.27(b), respectively). Only for some scenarios, two vehicles at a maximum were blocking, respectively one agent was superfluous (in the SAPg series).

8.3.4 Analysis of Results

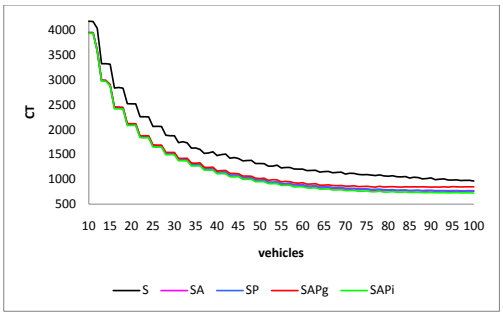
In order to prove that the IBC approach represented by the DIC model and instantiated by the PIC mechanism allows for the design of more efficient self-organizing emergent solutions as well as solution processes, we have to analyze the experimental results described in the last subsection in more detail. In particular, we have to evaluate the two dimensions of efficiency mentioned in the introduction (see Section 1.1): The efficiency of the solution produced by the self-organizing emergent system coordinating by means of PIC, i. e. mainly the required costs and time, and the efficiency of the solution process itself, i. e. the costs for producing the solution in terms of messages sent respectively infochemicals generated. Equation 8.1 defines an absolute measure for the solution efficiency eff_{sol}^{abs} in an experiment series es.

$$eff_{sol}^{abs}(es) = \frac{CT^{-1}(es)}{TTC(es)} \tag{8.1}$$

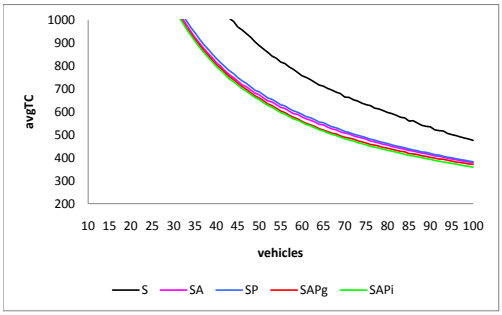
This measure correlates the two measures TTC and CT. Analyzing only one of the two is not enough to make a point regarding the solution efficiency. For instance, although in all experiment series that coordinate by means of more than one type of infochemical the TTC are lower than the TTC in the S series, this could be achieved at the cost of higher CTs, e. g. if only one vehicle is used for the solution. In turn, a lower CT could be achieved at the cost of higher TTC, e. g. if more vehicles are



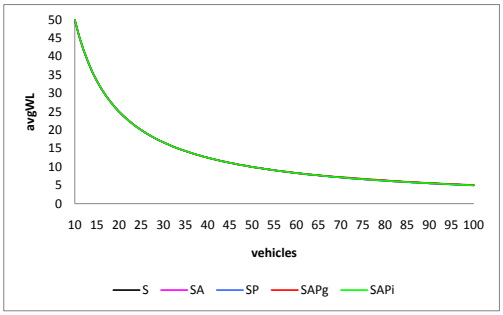
(a) Total travel costs



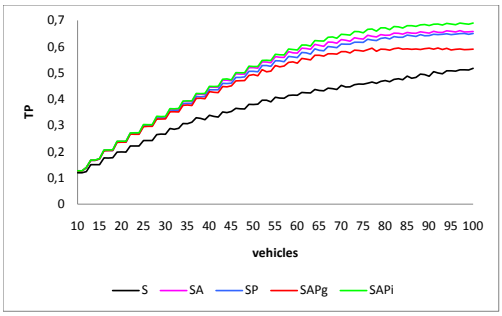
(b) Completion time



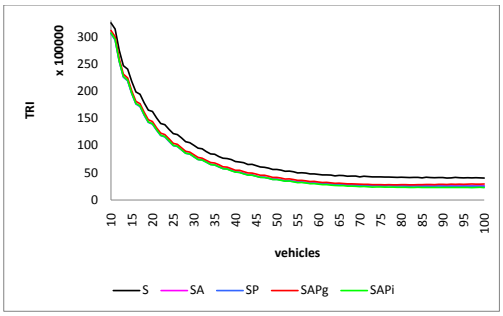
(c) Average travel costs per vehicle



(d) Average workload per vehicle

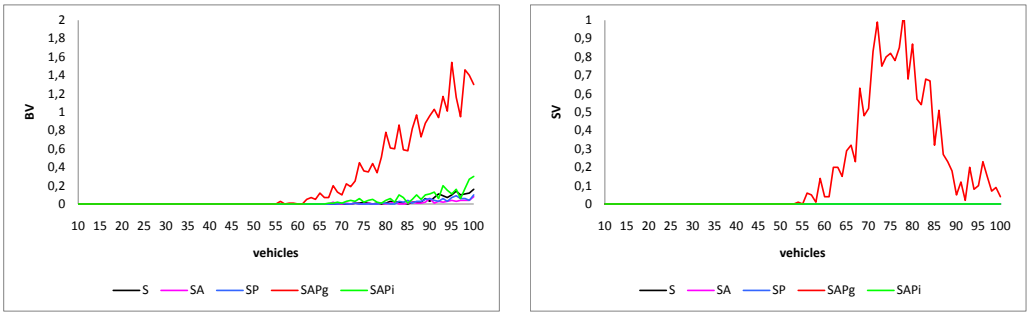


(e) Throughput



(f) Total rate of infochemicals

Figure 8.26: Experimental results for 500 transportation requests in *AM-Env*



(a) Blocking vehicles (b) Superfluous vehicles

Figure 8.27: Extended results for 500 transportation requests in *AM-Env*

used for the solution. Thus, eff_{sol}^{abs} is defined by the relation of the reverse CT to the TTC of the solutions in an experiment series. Apparently, a higher CT will decrease the efficiency, whereas lower TTC will increase the efficiency. Because we are interested in a relative measure, Equation 8.2 defines the solution efficiency eff_{sol} by correlating eff_{sol}^{abs} with the respective absolute solution efficiency of the S series. This provides us with a percentage, how efficient the coordination by means of different types of infochemicals is compared to the coordination by means of only one type of infochemical.

$$eff_{sol}(es) = eff_{sol}^{abs}(es) \cdot eff_{sol}^{abs}(S)^{-1} - 1 \tag{8.2}$$

However, the solution efficiency taken by itself does not give an indication on the costs of producing this solution, i. e. how many infochemicals have to be generated in order to produce an efficient solution. Thus, Equation 8.3 defines the measure for the absolute efficiency of the solution process eff_{pro}^{abs} in an experiment series.

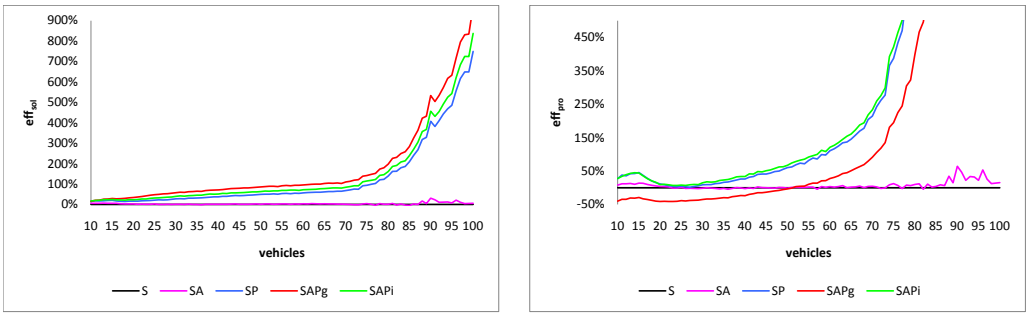
$$eff_{pro}^{abs}(es) = \frac{eff_{sol}^{abs}(es)}{TRI(es)} \tag{8.3}$$

This measure correlates the absolute solution efficiency eff_{sol}^{abs} with the respective TRI measure. Again, because we are interested in a relative measure, Equation 8.4 defines the solution process efficiency eff_{pro} by correlating eff_{pro}^{abs} with the respective absolute solution process efficiency of the S series. This provides us with a percentage, how efficient the solution process by means of different types of infochemicals is compared to the efficiency of the solution process by means of only one type of infochemical.

$$eff_{pro}(es) = eff_{pro}^{abs}(es) \cdot eff_{pro}^{abs}(S)^{-1} - 1 \tag{8.4}$$

Figures 8.28 – 8.42 depict both the solution efficiency and solution process efficiency for each of the PDP instances we have experimented with.

According to Figure 8.28(a), solutions based on multiple infochemicals, in particular in the *SP*, *SAPi*, and *SAPg* series, are much more efficient (up to 837%) for 100 transportation requests in the smallest environment *TW-Env 1* than the solutions based on synomones only in the *S* series, which represent the base line. Whereas the additional use of allomones (in *SA*) achieves only marginal improvements compared the the *S* series, the additional use of pheromones (in *SP*, *SAPi*, and *SAPg*) in contrast improves the efficiency of the solution significantly. The reason is that allomones only improve the existing interaction between station agents and vehicle agents, whereas pheromones in contrast allow for new interactions between the vehicles themselves, which improves the efficiency. Furthermore, whereas the coordination by different types of infochemicals using the same dynamics (such as in *SAPg*) is able to achieve the best efficiency of all solutions, mostly because information is exchanged more frequently, this relation changes for the solution process efficiency (see Figure 8.28(b)). Here, the efficiency of the *SAPg* series is up to 51 vehicles actually inferior to the efficiency of the *S* series. Only then, the efficiency of this solution process is better than the *S* and *SA* series. However, the best solution process efficiency can be achieved with the individual configuration of different types of infochemicals (in *SAPi*, up to 8119%), in other words infochemicals with different functions, dynamics, and semantics as provided by the DIC model.



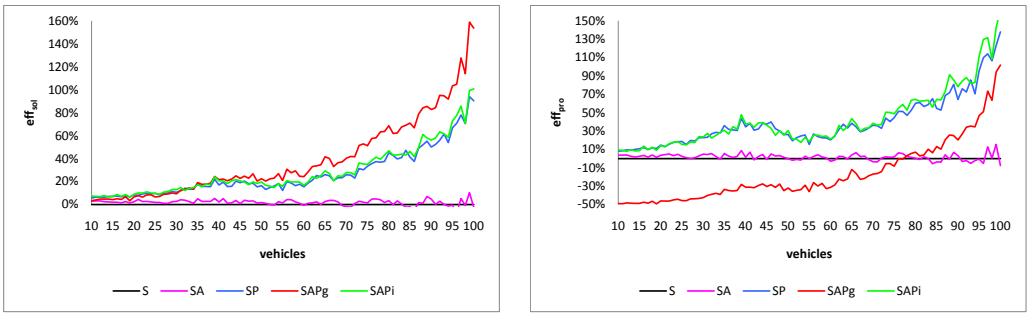
(a) Solution efficiency

(b) Solution process efficiency

Figure 8.28: Efficiency evaluation for 100 transportation requests in *TW-Env 1*

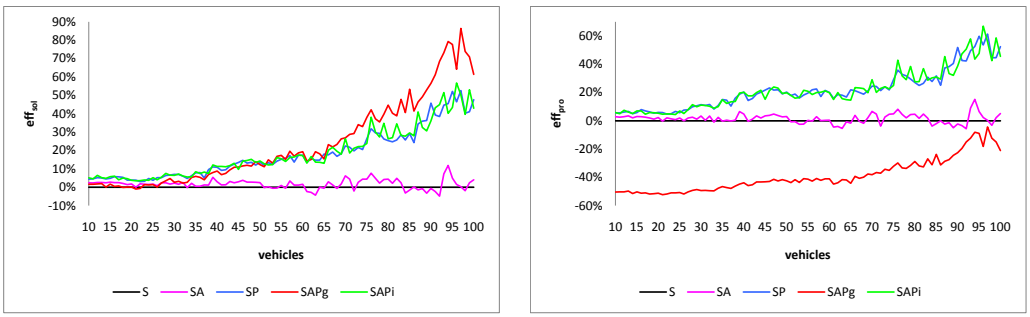
When considering 250 respectively 500 transportation requests in the same environment (see Figure 8.29 respectively Figure 8.30), the results are quite similar. The *SAPg* series predominantly provides the best solution efficiency (see Figure 8.29(a) respectively Figure 8.30(a)), even though the achieved improvement is lower than for 100 transportation requests (up to 159% respectively 86%). However, for 250 transportation requests, the SAPi series achieves a better solution efficiency for up to 31 vehicles, for 500 transportation requests the solution efficiency is even better for up to 51 vehicles. Whereas the solution process efficiency of the *SAPi* series is again the best for almost all numbers of vehicles, the *SAPg* series improves the solution process efficiency of the *S* series only for 76 vehicles and higher in the case of 250 transportation requests, while the efficiency is not improved at all for 500

transportation requests. The reason for this worsening with an increasing amount of transportation requests is the huge number of infochemicals that has to be generated, if the dynamics of all infochemicals are the same. A similar reason justifies the rare peaks in which the *SP* series outperforms the *SAPi* series. For some PDP instances the coordination by synomones and pheromones is already sufficient enough, so that the additional emission of allomones generates only communication overhead.



(a) Solution efficiency (b) Solution process efficiency

Figure 8.29: Efficiency evaluation for 250 transportation requests in *TW-Env 1*

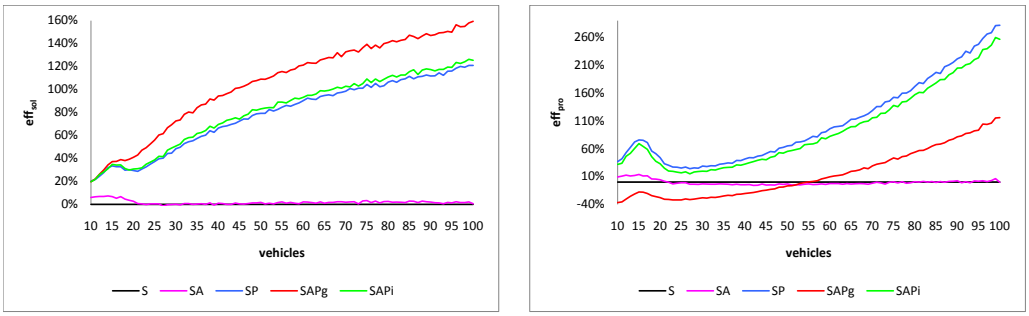


(a) Solution efficiency (b) Solution process efficiency

Figure 8.30: Efficiency evaluation for 500 transportation requests in *TW-Env 1*

Whereas the trend lines of the relative solution (process) efficiency provided by the *SP*, *SAPi*, and *SAPg* series in the environment *TW-Env 1* are nearly steadily increasing, for the larger environments *TW-Env 2* (see Figure 8.31, Figure 8.32, and Figure 8.33), as well as later *TW-Env 3* and *TW-Env 4*, we begin to observe local maxima in these series, after which the efficiency improvements decrease for a certain number of vehicles. The reasons for these maxima are obviously located in the TTC, CT, and TRI results. In more detail, the number of vehicles for that the CT or the TRI are minimal, is not the same number of vehicles for that the TTC reach a constant level respectively change their slope. Furthermore, the *SP*, *SAPi*, and *SAPg* series have no linear relationship to the S series. For instance, as

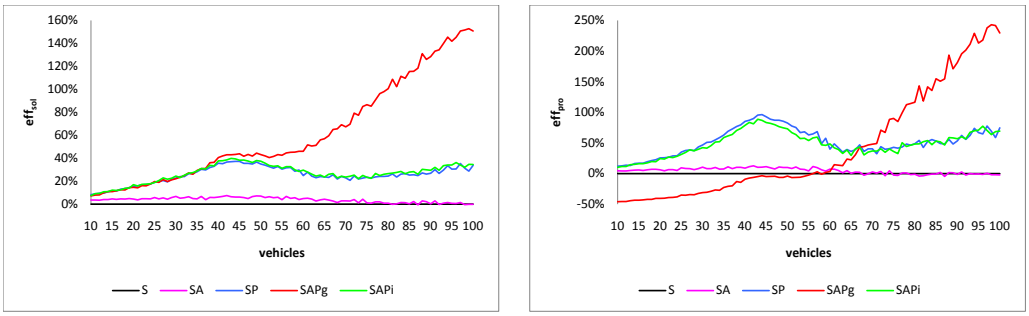
depicted in Figure 8.13(b) the S series achieves the lowest CT with 34 vehicles, the *SP*, *SAPi*, and *SAPg* series have reached their constant level already at around 22 vehicles. Thus, apart of the TTC, the CT of the *S* series improves the efficiency between 22 and 34 vehicles compared to the *SP*, *SAPi*, and *SAPg* series, which is why the latter series have a negative slope around these numbers of vehicles.



(a) Solution efficiency (b) Solution process efficiency

Figure 8.31: Efficiency evaluation for 100 transportation requests in *TW-Env 2*

The *SP* series is able to generate mostly the highest solution process efficiency at least for 100 and 250 transportation requests (see Figure 8.31(b), and Figure 8.32(b)), whereas it is nearly equal to the process efficiency of the *SAPi* series for 500 transportation requests (see Figure 8.33(b)). Again, for some of these PDP scenarios the coordination by synomones and pheromones is sufficient enough.

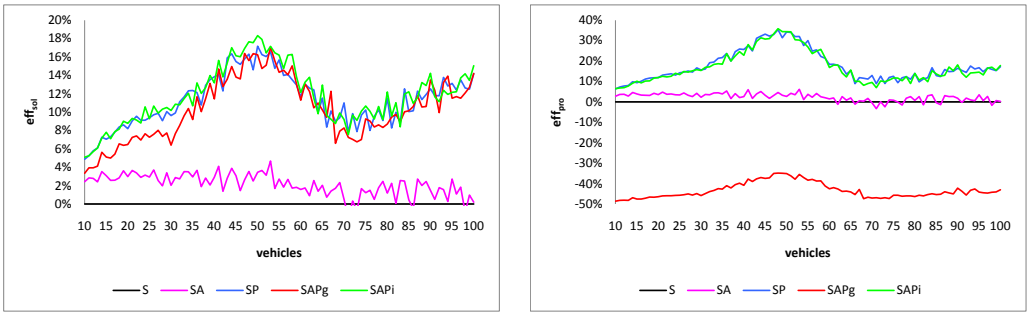


(a) Solution efficiency (b) Solution process efficiency

Figure 8.32: Efficiency evaluation for 250 transportation requests in *TW-Env 2*

Whereas these two series never worsen the solution process efficiency compared to the *S* series, the *SAPg* series does. Only for 250 transportation requests the SAPg series improves the efficiency significantly starting with 55 vehicles and higher. The reason is that the *SAPg* series is able to achieve nearly constant TTC and CT for this PDP scenario compared with all other series (see Figure 8.14).

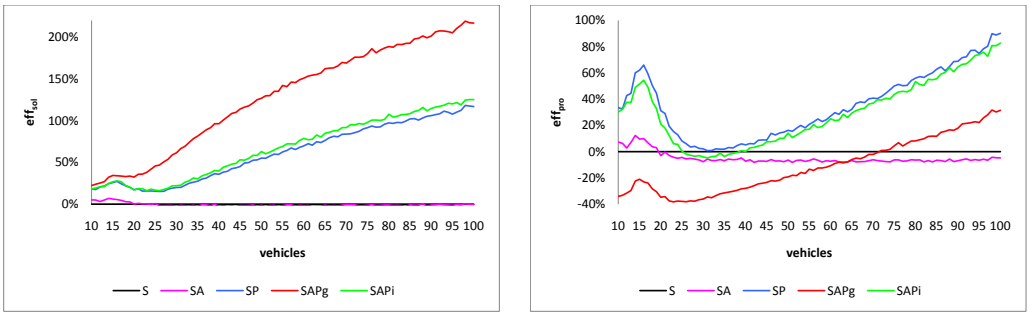
For the environment *TW-Env 3* the experiment series produce a quite similar



(a) Solution efficiency (b) Solution process efficiency

Figure 8.33: Efficiency evaluation for 500 transportation requests in *TW-Env 2*

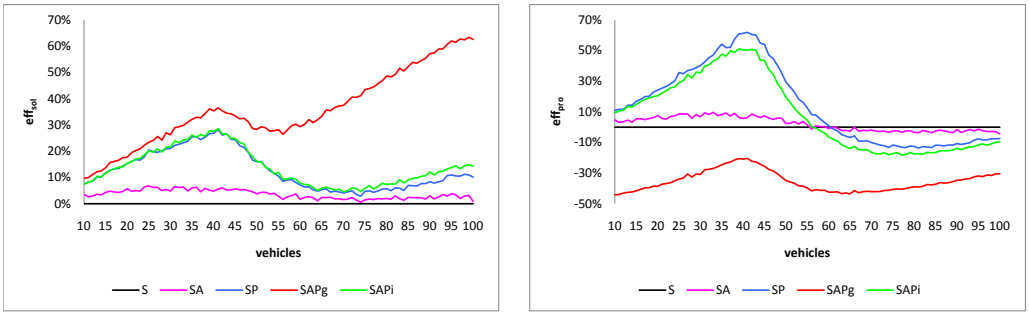
efficiency result compared with the environment TW-Env 2 (see Figure 8.34, Figure 8.35, and Figure 8.36). However, an interesting phenomenon can be observed in Figure 8.34(b) and Figure 8.35(b). Although the solution efficiency of the *SAPi* and *SP* series for 100 and 250 transportation requests is better than in the *S* series, the solution process efficiency in the *SAPi* and later in the *SP* series as well for a certain period is lower than in the *S* series. In other words, the communication costs for producing a more efficient solution in these periods are very high.



(a) Solution efficiency (b) Solution process efficiency

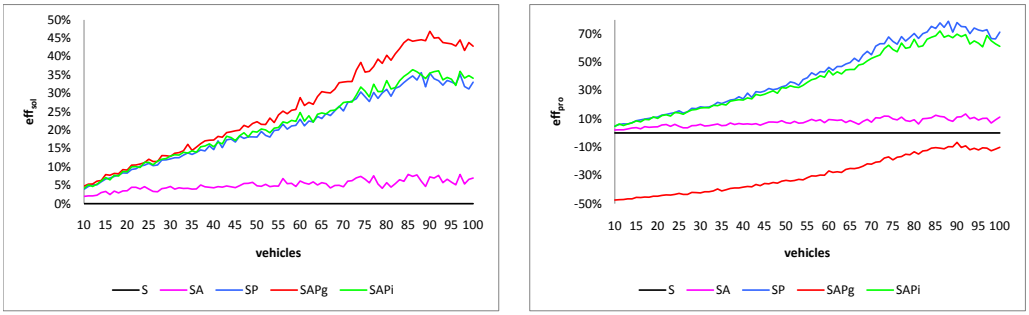
Figure 8.34: Efficiency evaluation for 100 transportation requests in *TW-Env 3*

With regard to the solution process efficiency for 500 transportation requests (see Figure 8.36(b)) we cannot observe this phenomenon anymore. Here, the solution process efficiency of the respective two series never worsens the efficiency of the *S* series. However, we again observe this phenomenon for 100 and 250 transportation requests even in the largest environment *TW-Env 4* (see Figure 8.37(b) and Figure 8.38(b)). Similarly, the solution efficiency (see Figure 8.37(a) and Figure 8.38(a)) can be improved but the solution process efficiency worsens for a certain period. For 500 transportation requests (see Figure 8.39) again both the solution efficiency and solution process efficiency is improved by these series as well.



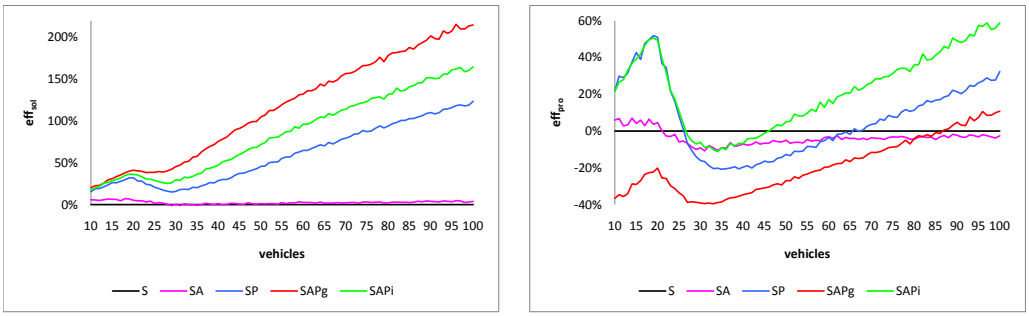
(a) Solution efficiency (b) Solution process efficiency

Figure 8.35: Efficiency evaluation for 250 transportation requests in *TW-Env 3*



(a) Solution efficiency (b) Solution process efficiency

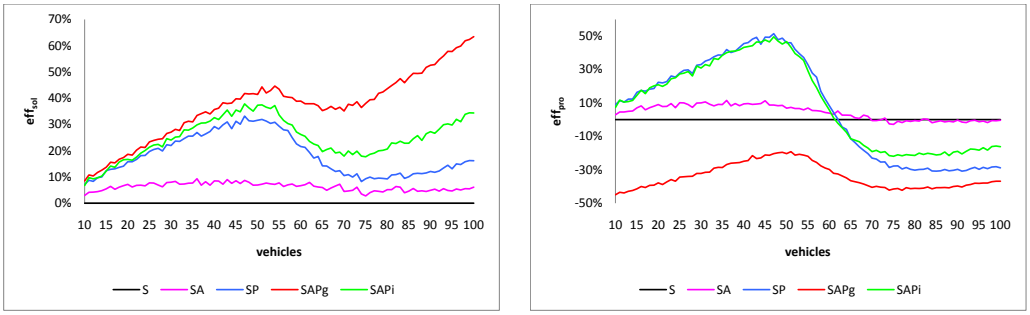
Figure 8.36: Efficiency evaluation for 500 transportation requests in *TW-Env 3*



(a) Solution efficiency (b) Solution process efficiency

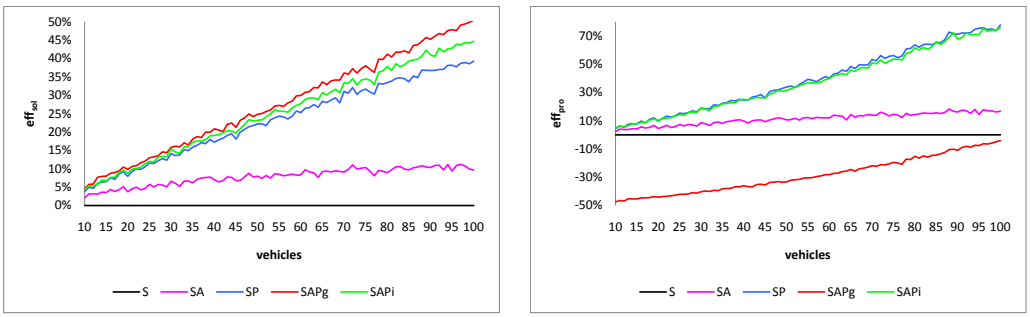
Figure 8.37: Efficiency evaluation for 100 transportation requests in *TW-Env 4*

With regard to the second case study, the solution efficiency was improved at most by the *SAPi* series, followed for almost all scenarios by the *SAPg*, *SP*, and *SA* series (see Figure 8.40(a), Figure 8.41(a), Figure 8.42(a)). Only for 500 transportation request, the improvements by the *SA* series are better then by the *SP* and *SAPg* series. Similarly, the solution process efficiency could be improved at most by the SAPi series as well (see Figure 8.40(b), Figure 8.41(b), Figure 8.42(b)), whereas the solution process efficiency improvement of all other series varies.



(a) Solution efficiency (b) Solution process efficiency

Figure 8.38: Efficiency evaluation for 250 transportation requests in *TW-Env 4*

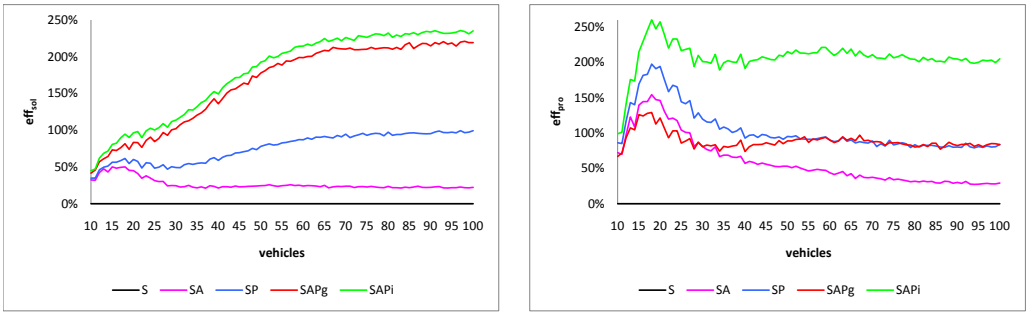


(a) Solution efficiency (b) Solution process efficiency

Figure 8.39: Efficiency evaluation for 500 transportation requests in *TW-Env 4*

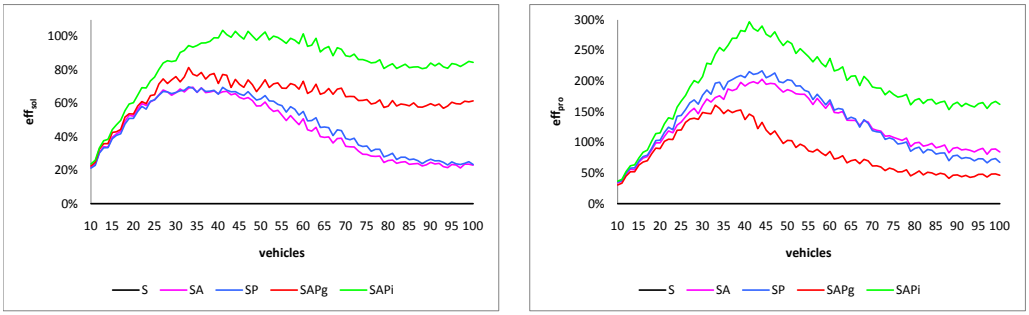
8.4 Experiments Regarding the EIA Approach

To evaluate the usefulness of the EIA approach, we have performed several experiments with the instantiation of the EIA described in Section 7.3. Creating experimental instances of a PDP(TW), in more detail sequences of run instances, that allow an appropriate evaluation is not a straightforward task, since the evaluation requires sequences that on the one hand include some recurring tasks, but on



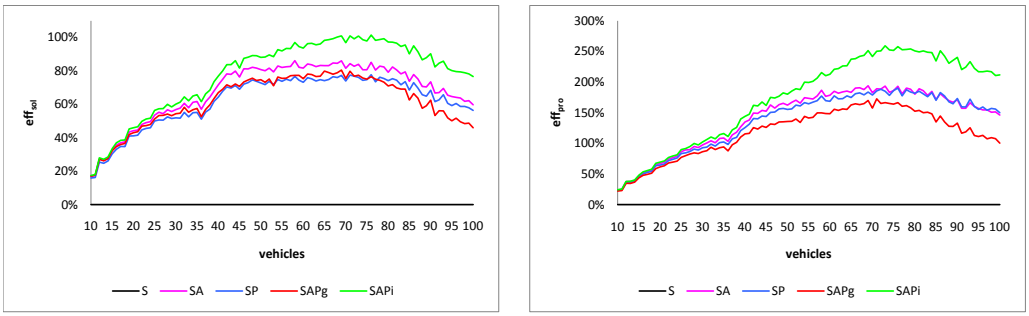
(a) Solution efficiency (b) Solution process efficiency

Figure 8.40: Efficiency evaluation for 100 transportation requests in *AM-Env*



(a) Solution efficiency (b) Solution process efficiency

Figure 8.41: Efficiency evaluation for 250 transportation requests in *AM-Env*



(a) Solution efficiency (b) Solution process efficiency

Figure 8.42: Efficiency evaluation for 500 transportation requests in *AM-Env*

the other hand also have enough randomness to allow the argument that the EIA approach will work for many problem instances.

8.4.1 Experiment Preparation

To test and evaluate the capabilities of the EIA approach, we thus have created three kinds of scenarios:

- **Crafted scenarios:** The first kind of scenarios was crafted (indicated by scenario names starting with "craft-...", see Table 8.5), i. e. the layout of the stations and the order of the tasks were intended to provoke a situation in which the solution of the basic system coordinating by means of PIC without any advice by the EIA is known to be suboptimal.
- **Random scenarios:** The second kind of scenarios contains a randomly created set of recurring tasks (indicated by scenario names starting with "rand-..."), in order to validate the EIA approach also for scenarios that have not been devised to explicitly provoke bad behavior. Even the distribution of the pickup/delivery stations over the environment was random. These scenarios were created by an application that outputs randomized scenarios with given input parameters (such as the number of stations, the size of the environment, and number of tasks).
- **Changing scenarios:** Because an important aspect of real world PDPs is that the set of recurring tasks may change over time, the third kind of scenarios contains changing recurring tasks (indicated by scenario names starting with "chang-..."), i. e. the set of recurring tasks is not constant as in the previous two kinds of scenarios, but changes over time. This continuously forces the EIA to adapt its advice to new situations.

Scenarios of any of the three kinds may additionally contain fixed time windows (indicated by scenario names ending with "...-TW"), in which the tasks have to be fulfilled. If a solution violates the time windows, the goal function will punish this behavior severely. Thus, in a good solution, the tasks are either fulfilled within the time windows or shortly after in case that it is not possible to stay within the time windows. In the following, we explain the created scenarios in more detail, starting with the scenarios with not-changing sets of recurring tasks, i. e. crafted scenarios and random scenarios, and afterwards scenarios with changing recurring tasks.

8.4.1.1 Scenarios with Not-Changing Recurring Tasks

Table 8.5 gives an overview of the properties of the different created scenarios with not-changing recurring tasks regarding environment factors. The smaller scenarios contain 4 respectively 6 recurring tasks (also indicated by the Arab number in the scenario names) and a total of 7 pickup/delivery stations. They were performed on

an environment consisting of a grid of 11 × 11 locations, where each location has connections to its direct neighbor (see for instance Figure 8.43). The scenarios with 8 recurring tasks were performed on an environment of 21 × 21 locations with a total of nine stations. All scenarios contain two vehicles to fulfill the tasks and one depot, at which the vehicles are housed. The depot is always set in the middle of the environment.

Scenario	Recurring tasks total	Recurring tasks probability	Stations	Depot	Vehicles	Map size
craft-4-I	4	1.00	7	1	2	11x11
craft-4-I-TW	4	1.00	7	1	2	11x11
craft-6-I	6	0.95 - 1.00	7	1	2	11x11
craft-6-I-TW	6	0.95 - 1.00	7	1	2	11x11
rand-6-I	6	0.95	7	1	2	11x11
rand-6-II	6	0.95	7	1	2	11x11
rand-8-I	8	0.95	9	1	2	21x21
rand-8-II	8	0.95	9	1	2	21x21
craft-8-I	8	0.95	9	1	2	21x21

Table 8.5: Properties of scenarios with not-changing recurring tasks regarding environment factors

The essential key for the creation of generally valid scenarios was the introduction of randomness. Just as in real world PDPs, there are various influences that alter the set of recurring tasks to be fulfilled every day. Even though most tasks of the set of recurring tasks do appear every day, there may be days at which the one or the other task does not appear. This is modeled by a probability that is assigned to each task. As depicted in Table 8.5, each of the recurring tasks (or a slight modification of it within *sim*) appears in each run instance of the corresponding scenario with a given probability of at least 95%, except for the craft-4-I-. . . scenarios, where the probability was 100%. In addition, there was "noise" in each run instance, i. e. randomly generated tasks that can occur at any point in time during a run instance. These two factors make even predefined scenarios very general and realistic.

Table 8.6 gives an overview of the properties of the different created scenarios with not-changing recurring tasks regarding runs and tasks. The length of the sequence of run instances that forms a scenario depends on the number of intended recurring tasks, namely 10 run instances for scenarios with 4 recurring tasks, 20 run instances for scenarios with 6 recurring tasks, and 40 run instances for scenarios with 8 recurring tasks. To give the EIA the chance to identify the these recurring tasks, a number of training runs are necessary, after which the optimization can start. The concrete number of training runs varies between 3 and 12 for the scenarios. The number of noise tasks was chosen randomly from between 10 to 30 percent of

the number of intended recurring tasks. Please note, that the displayed values for the tasks are rounded average values from three instances of each scenario. In the following we explain each of the scenarios listed above in more detail.

Scenario	Runs		Tasks		Noise
	total	training	total	noise	(in percent)
craft-4-I	10	3	53	13	25.00%
craft-4-I-TW	10	3	54	14	26.38%
craft-6-I	20	6	150	35	23.50%
craft-6-I-TW	20	6	159	43	26.89%
rand-6-I	20	3	131	17	13.01%
rand-6-II	20	3	131	15	11.73%
rand-8-I	40	12	409	91	22.33%
rand-8-II	40	12	393	89	22.71%
craft-8-I	40	12	393	73	18.59%

Table 8.6: Properties of scenarios with not-changing recurring tasks regarding runs and tasks

- **craft-4-I:** This scenario defines a basic set of tasks that is solved poorly by the basic system on its own and that hence forms the set of recurring tasks the EIA should identify. Most of the following scenarios, in particular all crafted scenarios, are based on this minimal set and are augmented with additional tasks that repeat this pattern again. Table 8.7 lists the tasks that appear in this scenario, whereas Figure 8.43 exemplary illustrates the distribution of the pickup/delivery stations and the depot in this scenario over the environment.

	Pickup		Delivery		Load size	Probability
	station	time	station	time		
Task 1	4	25	6	30	20	1.00
Task 2	3	25	5	25	20	1.00
Task 3	2	50	7	55	20	1.00
Task 4	1	50	7	55	20	1.00

Table 8.7: Tasks of the scenario *craft-4-I*

According to the task set listed in Table 8.7, both vehicles first move to the lower right corner to perform tasks 1 and 2. Subsequently, they move to the upper left corner of the environment to perform tasks 3 and 4. Thus, both vehicles move almost across the entire environment, generating high travel costs. Additionally, tasks 3 and 4 can not be serviced right away as the vehicles

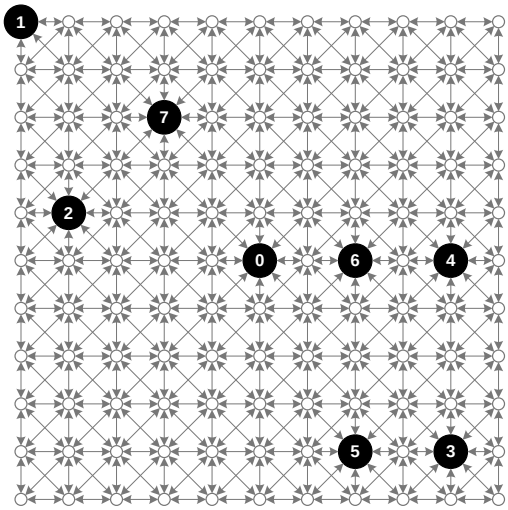


Figure 8.43: Distribution of stations and depot over an 11×11 environment for the scenario *craft-4-I*

first have to move there after they perceived them in iteration 50. A better strategy with regard to TTC would have only one vehicle go down into the lower right corner performing tasks 1 and 2, while the other vehicle waits in the depot until tasks 3 and 4 are announced and then performs those two tasks. This is exactly the solution the EIA will create.

- **craft-4-I-TW:** Employing the same tasks as in the *craft-4-I* scenario above, this scenario additionally enforces time windows in which the tasks have to be fulfilled. As mentioned above, this in turn limits the number of good solutions severely as only those solutions that stay within the time windows yield good evaluations.

- **craft-6-I:** Two additional tasks were added to the recurring tasks of scenario *craft-4-I* that cause the basic system to perform in a suboptimal way twice during a run. In addition to the problem instance described above, the vehicles now have to move back to the lower right corner of the environment and do so – when unadvised – in the wrong order. The EIA's advice will cause the vehicle that solved the tasks in the lower right corner to serve the tasks that are appearing at a later point in time as well.

- **craft-6-I-TW:** Again, the same tasks are employed as in the *craft-6-I* scenario described above but with enforced time windows. However, in this scenario the first four tasks appear in every run instance, whereas the last two only appear with a probability of 95%. The time windows are relatively small, forcing the EIA to create a solution that accomodates the times. Table 8.8 lists the tasks of this scenario.

	Pickup		Delivery		Load size	Probability
	station	time	station	time		
Task 1	4	0-8	6	0-14	20	1.00
Task 2	3	0-8	5	0-12	20	1.00
Task 3	2	25-41	7	30-41	20	1.00
Task 4	1	25-41	7	30-41	20	1.00
Task 5	6	50-56	4	50-60	20	0.95
Task 6	5	52-58	3	52-62	20	0.95

Table 8.8: Tasks of the scenario *craft-6-I-TW*

- **rand-6-I:** This purely randomly generated scenario has an even distribution of seven nodes on a 11×11 environment and requires the vehicles to serve six tasks that are created randomly as well. That means that the locations for pickup and delivery as well as the starting times and the load sizes are random values that were assigned at the time the scenario was created. The only constraint was that the timing of the pickup and the delivery are reasonably close together as not to have vehicles pick up goods and not let them see where they have to be brought to.

- **rand-6-II:** A scenario that uses the same environment as the one above but incorporates different, randomly generated tasks.

- **rand-8-I:** This scenario distributes eight tasks on an environment of 21×21 locations. As with the smaller randomly generated scenarios, the tasks are random with regard to their location, load size, and timing.

- **rand-8-II:** This scenario was created with the same input to the randomizer as *rand-8-I* but with other, randomly generated tasks. While the environment is the same, this scenario shows different characteristics with regard to timing and location of tasks.

- **craft-8-I:** This scenario is based on the random task set of scenario *rand-8-I* but has been manually altered so that two tasks are very similar to each other, thus being put into the same cluster when identifying the recurring tasks. This will demonstrate the ability of the EIA to cope with situations in which very similar tasks occur in the system that are part of the set of recurring tasks but not distinguished by the clustering algorithm.

8.4.1.2 Scenarios with Changing Recurring Tasks

In contrast to the aforementioned scenarios where the recurring tasks do not change, in the subsequent scenarios the recurring tasks change at some point in time. Please note, only the transportation requests change, while the distribution of the stations

and the layout of the environment remains the same. In these scenarios, the EIA initially has to create advice for the recurring tasks it identifies in a first block of run instances. Then, the recurring tasks change in the following block of run instances, forcing the EIA to adapt to the new situation and to create advice for the basic system. Usually, there is again a learning phase in which the recurring tasks are extracted from the agent data over a couple of run instances. After this learning phase, the EIA starts to create advice for the new set of recurring tasks. If the situation changes again, the EIA has to adapt again and so forth.

Table 8.9 gives an overview of the properties of the different created scenarios with changing recurring tasks regarding environment factors. All created scenarios contain 6 recurring tasks and again a total of 7 pickup/delivery stations. They are again performed on an environment consisting of a grid of 11×11 locations with two vehicles to fulfill the tasks and one depot, set in the middle of the environment. Each of the recurring tasks (or a slight modification of it within *sim*) appears in each run instance of the corresponding scenario with a given probability of at least 95% as well.

Scenario	**Recurring tasks**		**Stations**	**Depot**	**Vehicles**	**Map size**
	total	**probability**				
chang-6-I	6,6	0.95 - 1.00	7	1	2	11x11
chang-6-II	6,6	0.95 - 1.00	7	1	2	11x11
chang-6-III	6,6,6	0.95 - 1.00	7	1	2	11x11

Table 8.9: Properties of scenarios with changing recurring tasks regarding environment factors

Table 8.10 gives an overview of the properties of the different created scenarios with changing recurring tasks regarding runs and tasks. Contrary to the scenarios with not-changing recurring tasks, the length of the sequence of run instances that forms a scenario now is different. The number of training runs remains constant at 6 for all scenarios. The number of noise tasks was chosen randomly around 11 percent of the number of intended recurring tasks. Please note again, that the displayed values for the tasks are rounded average values from three instances of each scenario. In the following we explain each of the scenarios listed above in more detail.

- **chang-6-I:** This scenario starts with 20 run instances using one set of 6 recurring tasks. The instances were created with additional random tasks and probabilities for the recurring tasks as described before. Then follow 12 run instances created using a different randomly created set of 6 recurring tasks, again with random influences during runtime, after which follow another 20 run instances with the first set of recurring tasks. The number 12 was chosen, because it is just too small to allow for a change in advice by the EIA.

Scenario	Runs		Tasks		Noise
	total	training	total	noise	(in percent)
chang-6-I	52	6	344	43	12.61%
chang-6-II	64	6	417	53	12.70%
chang-6-III	60	6	377	37	9.72%

Table 8.10: Properties of scenarios with changing recurring tasks regarding runs and tasks

- **chang-6-II:** For this scenario, the same sets of recurring tasks have been used as in scenario *chang-6-I*, but now having 24 instead of 12 run instances before changing "back". This should allow for a change in advice by the EIA.
- **chang-6-III:** This scenario is completely different from the first two and has been randomly generated. It has three blocks of run instances using three different (random) sets of 6 recurring tasks, each set for 20 run instances.

8.4.2 Experiment Execution

Admittedly, the scenarios that were created for the evaluation of the EIA approach are quite simple compared to the scenarios created for the evaluation of the IBC approach. However, they clearly show the potential of the EIA approach. More advanced scenarios could not be used due to the simple nature of the optimization algorithm within the EIA that was employed to calculate optimal solutions. With a more sophisticated optimization algorithm, solutions for larger problems could be calculated in an acceptable time.

To be able to compare the results of the system running without advice and with advice by the EIA, the first step was to prepare a repeatable execution of the created scenarios, which allows for a direct and fair comparison of the system performance with and without the EIA. Therefore, for each of the created scenarios, an environment and a scenario configuration file were created as described in Section 8.1.2. Then a simulation of each scenario was performed and recorded, in which noise tasks were created and the task occurrence probability was considered. The recording generates a new scenario configuration file that can then be used to rerun the simulation exactly the way it was at the time of recording, i. e. all the tasks appear with a probability of 100% and no additional noise tasks are created. To overcome statistical anomalies resulting from the incorpoprated randomness, each of the scenarios defined in the last subsection has been simulated and recorded three times, resulting in three different scenario configuration files. Every recording was then replayed, once without the EIA and once with it, while both replays generate exactly the same tasks.

The number of parameters that can be changed in the EIA is relatively small. Most of the parameters have been set to sensible default values, which turned out to

be sufficient in all of the regarded cases. One important exemption are the penalty multipliers of the quality function of the optimization algorithm, which have been carefully evaluated. Subtle changes in these factors can change the solutions the algorithm creates and may lead to later bounding and therefore longer runtime. The values presented here reflect a good compromise that let the algorithm generate solutions that are both reasonable to an experienced human observer and keep the runtimes in check. Table 8.11 shows the most important parameters and the values that were used in the evaluations. The infochemical parameters were left untouched.

Name	Description	Value
Clustering and Optimization		
k_{max}	time window for clusterer	20
qualthresh	threshold to determine if emergent solution should be optimized	95%
minocc	runs a task has to appear in to be recurring	0.7
clustthresh	threshold to determine if a task belongs to a cluster	20
Distance function for clustering of tasks		
τ	weighing factor for locations	0.3
υ	weighing factor for goods	0.1
ω	weighing factors for time windows	0.3
Application of rules		
synthresh	maximal distance of abstracted synomone to perceived synomone	20
	iterations for which a rule is applied consecutively	100
Quality function of optimization algorithm		
ϖ	weight parameter to end up values higher than 1	10^5
λ	penalty multiplier for distance covered by agents	1
μ	penalty multiplier for incorrect order in solution	15
ν	penalty multiplier for violations of time window, if any	3
	when no times windows are present	0

Table 8.11: Parameter values for the evaluation of the EIA approach

To get a performance measure for the solutions, the quality function of the EIA was used. Its value depends on the distance traveled by the vehicles, the task order, and the violation of time windows (see equation 7.20). The system that performed better according to this quality function got a higher value for this run than the other system.

To compare the solutions for the entire scenario, the solutions of all runs are averaged. Table 8.12 gives an example of the data collected during the simulation of scenario *craft-4-I*. As defined in Table 8.6, a total of three training runs were performed first before the optimization process started. Additionally, the example

shows that the solution of the recurring tasks continuously converges towards the optimum. A thorough examination of the data collected and an explanation of the data is given in Subsection 8.4.4.

Run	Unoptimized solution (all tasks)	Enhanced solution (all)	Enhanced solution (recurring)	Optimal solution (recurring)	Rule created
1	1315.79	1315.79	1315.79		
2	1388.89	1388.89	2127.66		
3	2083.33	2083.33	2083.33		
4	943.400	943.400	2083.33	2777.78	TRUE
5	1265.82	1333.33	2083.33	2777.78	TRUE
...	...	...	...	...	...
9	1086.96	1315.79	2777.78	2777.78	
10	1282.05	1851.85	2777.78	2777.78	

Table 8.12: Shortened example of simulation data for *craft-4-I*

8.4.3 Experimental Results

Table 8.13 clearly shows that the EIA, even in its current, simple form, enhances the performance of the basic, self-organizing emergent system at runtime in each of the scenarios. Even those scenarios, which were created randomly, show an improvement. The results were compiled by averaging the results of all runs of all the experiments for the same scenario. The change between utilities is expressed in percent where a positive value means that the advised system performed better than the unadvised one.

As expected, the EIA is able to improve the solutions for the scenarios that were crafted to contain a set of recurring tasks that provokes suboptimal behavior. The solutions for the recurring set are often even optimal. The results clearly show that it is a valid assumption that optimization of a small recurring set leads to improvements even if the set itself is mixed with random tasks. Scenarios that only showed minimal improvements (such as *rand-8-II*) have already been solved in a very good way by the unadvised system. The impact of the EIA is therefore hardly noticeable. However, it is important to note that the EIA did not cause any deterioration – on the contrary, it even improved these near-optimal solutions.

While the improvements for scenarios that contain a set that is known to be solved in a suboptimal way are already promising, the improvements in the completely random scenarios are even more interesting. They show that the solutions of the basic system can also be improved under circumstances where such optimizations are not obvious for human observers. This promising result makes the EIA applicable to real world scenarios that consist of situations that are similar every day but contain additional, more or less random tasks interwoven in the recurring pattern.

Scenario	without EIA	with EIA	Improvement
craft-4-I	1319.37	1415.51	7.29%
cratf-4-I-TW	210.94	240.31	13.92%
craft-6-I	704.06	797.63	13.29%
craft-6-I-TW	140.46	170.41	21.32%
rand-6-I	627.26	737.41	17.56%
rand-6-II	566.05	726.69	28.38%
rand-8-I	177.96	186.05	7.49%
rand-8-II	181.08	185.41	2.39%
craft-8-I	244.17	250.90	2.76%
chang-6-I	836.45	956.74	14.38%
chang-6-II	800.74	936.90	17.00%
chang-6-III	728.74	874.22	19.96%

Table 8.13: Overall experimental results for the EIA approach

Additionally, the adaptability of the approach can be observed in the results for the scenarios with a changing task set. The EIA is capable to adapt to new scenarios even if these are fundamentally different from the ones before if given enough time to learn about the new scenario. In case the scenarios are not completely different and the rules that have previously been learned can still be applied, an advised system will perform even better as the existing advise improves the solution during the learning time for the new set.

Overall, the experimental evaluation shows that the EIA approach is able to provide the agents with good advice, without undermining the important and useful properties of the underlying self-organizing emergent system necessary to deal with the dynamic nature of the problem.

8.4.4 Analysis of Results

To show how the results were compiled, the process is illustrated by two simulations of the scenarios *craft-6-I* and *chang-6-III*. Table 8.14 shows a summary of the simulation of the scenario *craft-6-I* with and without the EIA. 151 tasks had to be executed during 20 runs. 37 of these tasks were noise tasks, i. e. were created randomly during the recording phase. This means that 114 tasks were part of recurring tasks in the respective runs. The optimal solution for the set of recurring tasks is 1449. This solution was reached for the recurring tasks within 12 runs, while 6 runs were attributed to the learning phase of the EIA and 6 rules were created. The unoptimized average solution for all tasks in a run was 711 while the average solution of the system with the EIA enabled was 812. This is an improvement of 14.25%.

There are several interesting things to note about these results. First of all, they show how the EIA optimizes the system for the recurring tasks. The system uses

Total number of tasks:	151
Number of noise tasks:	37
Noise percentage:	24.50%
Number of runs:	20
Optimal solution:	1449
Converged in runs:	12
Training runs:	6
Created rules:	6
Unoptimized solution (average):	711
Improved solution (average):	812
Improvement:	14.25%

Table 8.14: Results of an experiment for the scenario *craft-6-I*

the optimal solution for these tasks in the 12^{th} run. As the EIA did only start to optimize the system after the 6^{th} run, this means that it only required five runs to change the behavior of the agents in a way that lets it solve the set of recurring tasks in an optimal manner. Second of all, it becomes obvious that the optimization of the set of recurring tasks also leads to improvements in the overall solution if a lot of tasks that are not part of this set and even disturb the set are part of the runs. Almost one quarter of the tasks that were recorded were not part of the set of recurring tasks. Still, the advice created by the EIA had a significant positive impact on the results for all tasks.

The detailed data of the *craft-6-I* experiment is presented in Table 8.15. The data for the unoptimized and the improved solutions of the above table is deducted by averaging the second or third column respectively. The improved solution for the recurring tasks can be seen in the fourth column. A comparison with the fifth column – the optimal solution for this set of tasks – shows how the advice created by the EIA lets the agents perform better for this set and even reach the optimal solution in many cases, the first time in run 12. The fluctuations of the improved solutions are due to the random tasks that may change the order in which the recurring tasks are executed and the probabilities with which the recurring tasks are part of a run. To prevent the system from trying to optimize after runs that showed a great difference to the ones before due to interference by random noise (a form of over-optimization), solutions that are significantly worse than previous ones are not regarded (as in run 13 and 18). The table also shows that the improved solution is not always better than the unoptimized one, especially during the phase in which the EIA converges to the optimal solution for the recurring tasks (see e. g. run 10). Nonetheless, when regarded over several runs, the enhancements are significant and can be found in almost every run, even in those in which the set of recurring tasks was severely disturbed by random tasks (again runs 13 and 18).

Another interesting example is depicted in Table 8.16, which shows the data col-

Run	Unoptimized solution (all tasks)	Improved solution (all)	Improved solution (recurring)	Optimal solution (recurring)	Rule created
1	568.1818182	568.1818182	568.1818182		
2	1000.000000	1000.000000	1000.000000		
3	581.3953488	581.3953488	1000.000000		
4	694.4444444	694.4444444	1000.000000		
5	1388.888889	1388.888889	1111.111111		
6	343.6426117	343.6426117	1219.512195		
7	543.4782609	543.4782609	909.0909091	1449.275362	TRUE
8	427.3504274	467.2897196	909.0909091	1449.275362	TRUE
9	645.1612903	636.9426752	980.3921569	1449.275362	TRUE
10	751.8796992	617.2839506	1250.000000	1449.275362	TRUE
11	588.2352941	806.4516129	1282.051282	1449.275362	TRUE
12	602.4096386	699.3006993	1449.275362	1449.275362	
13	952.3809524	1136.363636	101.0101010		
14	800.0000000	980.3921569	1010.101010	1449.275362	TRUE
15	1000.000000	1449.275362	1449.275362	1449.275362	
16	613.4969325	884.9557522	1449.275362	1449.275362	
17	769.2307692	1123.595506	1449.275362	1449.275362	
18	1020.408163	1136.363636	102.0408163		
19	609.7560976	826.4462810	1449.275362	1449.275362	
20	316.4556962	358.4229391	1449.275362	1449.275362	

Table 8.15: Detailed results of an evaluation run for the scenario *craft-6-I*

lected during a simulation of scenario *chang-6-III*. The first 20 runs show a normal optimization process. Advice is created to accommodate the recurring tasks and the system reaches the optimal solution for the set of recurring tasks. In run 21 however, a completely different set of recurring tasks is presented to the system. The recurring tasks as seen by the EIA has not changed yet as the clusterer's time window still takes into account the previous runs. This situation changes after run 24 when the clusters are getting to small and the first old tasks are no longer part of the recurring tasks. Instead, the EIA either ignores the run due to an assumed statistical anomaly (run 24) or sees a significantly shortened set of recurring tasks (runs 25–27) with accordingly higher goal values. In runs 28–33 no set of recurring tasks could be found any more and the EIA therefore does not calculate the goal value of the emergent and optimal solutions. In run 34 there is finally enough data again to deduce a set of recurring tasks. However, as the goal value of the solution in run 27 was so high, the emergent solutions are still ignored as statistical anomalies. Finally, in run 36, the EIA has identified the recurring tasks again.

Run	**Unoptimized solution (all tasks)**	**Improved solution (all)**	**Improved solution (recurring)**	**Optimal solution (recurring)**	**Rule created**
1	465.1162791	465.1162791	465.1162791		
2	757.5757576	757.5757576	757.5757576		
3	746.2686567	746.2686567	1000.000000		
4	549.4505495	549.4505495	1000.000000	1449.275362	TRUE
5	1265.822785	1265.822785	1000.000000	1449.275362	TRUE
6	641.0256410	653.5947712	1010.101010	1449.275362	FALSE
7	598.8023952	613.4969325	1000.000000	1449.275362	TRUE
8	602.4096386	602.4096386	101.0101010		
9	574.7126437	724.6376812	1282.051282	1449.275362	TRUE
10	1000.000000	1250.000000	1250.000000	1449.275362	TRUE
11	781.25.0000	1149.425287	1449.275362	1449.275362	
12	763.3587786	869.5652174	101.0101010		
13	1388.888889	1694.915254	1449.275362	1449.275362	
14	666.6666667	909.0909091	1449.275362	1449.275362	
15	549.4505495	757.5757576	1449.275362	1449.275362	
16	364.9635036	487.804878	1449.275362	1449.275362	
17	854.7008547	1052.631579	1204.819277	1449.275362	FALSE
18	609.7560976	724.6376812	1449.275362	1449.275362	
19	568.1818182	613.4969325	101.0101010		
20	1388.888889	1694.915254	1449.275362	1449.275362	
21	751.8796992	826.4462810	97.37098345	1449.275362	FALSE
22	751.8796992	826.4462810	1000.000000	1449.275362	FALSE
23	990.0990099	1075.268817	877.1929825	1449.275362	FALSE
24	869.5652174	840.3361345	168.6340641		

Run	Unoptimized solution (all tasks)	Improved solution (all)	Improved solution (recurring)	Optimal solution (recurring)	Rule created
25	869.5652174	840.3361345	1612.903226	4166.666667	FALSE
26	751.8796992	826.4462810	4166.666667	4166.666667	
27	751.8796992	826.4462810	16666.66667	16666.66667	
28	751.8796992	826.4462810			
29	751.8796992	826.4462810			
30	990.0990099	1176.470588			
31	534.7593583	621.1180120			
32	751.8796992	826.4462810			
33	751.8796992	826.4462810			
34	751.8796992	826.4462810	1538.461538		
35	751.8796992	826.4462810	1052.631579		
36	689.6551724	826.4462810	826.4462810	854.7008547	
37	990.0990099	1176.470588	787.4015748	854.7008547	TRUE
38	751.8796992	763.3587786	763.3587786	854.7008547	TRUE
39	751.8796992	854.7008547	854.7008547	854.7008547	
40	990.0990099	1176.470588	787.4015748	854.7008547	FALSE
41	531.9148936	531.9148936	826.446281	854.7008547	
42	574.7126437	574.7126437	826.446281	854.7008547	
43	487.8048780	487.8048780	787.4015748	854.7008547	FALSE
44	574.7126437	574.7126437	1176.470588	1176.470588	
45	662.2516556	662.2516556	1123.595506	1176.470588	
46	574.7126437	574.7126437	1123.595506	1176.470588	
47	543.4782609	543.4782609	12500.00000	12500.00000	
48	574.7126437	574.7126437	5555.555556		
49	662.2516556	662.2516556	5555.555556		
50	574.7126437	574.7126437	5555.555556		
51	606.0606061	606.0606061	8333.333333	8333.333333	
52	1176.470588	1176.470588	5555.555556	8333.333333	TRUE
53	1176.470588	1562.500000	8333.333333	8333.333333	
54	574.7126437	719.4244604	4166.666667	4545.454545	TRUE
55	574.7126437	952.3809524	2777.777778	4545.454545	TRUE
56	346.0207612	378.7878788	1298.701299		
57	574.7126437	833.3333333	1149.425287		
58	574.7126437	833.3333333	1149.425287		
59	574.7126437	833.3333333	1149.425287	1298.701299	FALSE
60	800.0000000	1204.819277	1149.425287	1298.701299	TRUE

Table 8.16: Detailed results of an evaluation run for the scenario *chang-6-III*

The runs after run 40 show the same behavior. First, the EIA does not realize something has changed and still considers the old set of recurring tasks. Then, as the clusters contain less and less tasks, the set of recurring tasks gets smaller and goal values spike before the EIA has not enough data any more. This time however, the system identifies a part of the new recurring tasks in run 51 and starts optimizing for it. In run 54, another task is added to the new set of recurring tasks. The gaps in runs 56 – 58 can again be explained by the drastic change in goal values – interpreted as a statistical anomaly – as the EIA adds another task to the set of recurring tasks. It is only in run 59 that the EIA has finally found the correct 6 recurring tasks.

The reason the advised system performs so good in this scenario (an improvement of 14.23% in this simulation instance) is that the agents keep the rules the entire time. A rule that has been created for another set of recurring tasks can still be beneficial for another one. Additionally, the optimizations for the partial solutions in runs 52 – 55 enhance the solution of the emergent system already in an early stage of adaptation. The improvements would be even greater if the time window for the clusterer was narrower (allowing faster adaptation to a changing set) or each of the three blocks would be used for a longer time.

8.5 Conclusion

In this chapter we have proven that the models, mechanisms, and architectures developed in the previous chapters enable the design and operation of efficient self-organizing emergent systems. We have evaluated them experimentally in two realistic PDP case studies from the field of intralogistics, for which we have developed a simulation tool.

The experimental results and their analysis regarding the IBC approach have demonstrated that the decentralized coordination by means of multiple digital infochemicals with different functions, dynamics, and semantics within one coordination mechanism – as provided by the DIC model and instantiated by the *SAPi* experiment series of the PIC mechanism – is able to achieve a better solution regarding various quality measures. In particular with regard to the total travel costs (TTC), the completion time (CT), the throughput (TP), and the communication costs (TRI), which represent the most relevant quality measures, the improvements in comparison to the benchmarking *S* experiment series become very apparent. The significant reduction of TTC as well as TRI in various environments saves high operational expenditures for the operator of the self-organizing emergent system. The reduced CT as well as the increased TP at the same time increment the profit for the operating company. Furthermore, because the improved coordination in *SAPi* is able to adapt the number of used vehicles to the number of transportation requests to fulfill, it makes the use of a specific amount of vehicles superfluous, which additionally saves capital expenditures. In addition, the improved coordination reduces the number

of blocking vehicles, which is in particular necessary for small environments such as *TW-Env 1*, in which these vehicles congest the environment and thus worsen the solution. All in all, the experimental results thus not only have shown that the PIC mechanism scales with the number of vehicles, but also with the number of transportation requests to fulfill as well as the size of the environment.

The IBC approach furthermore allows for the design of self-organizing emergent system exhibiting both a high solution efficiency as well as a high solution process efficiency. The solution efficiency results of the *SAPg* series suggest the conclusion that if the communication costs are negligible (and therewith the solution process efficiency), all types of infochemicals should have to be propagated very frequently in a broadcast style over the entire environment to achieve the best efficiency. In particular the wide-range propagation of pheromones and their slow evaporation yields the highest solution efficiency in almost all *TW-Env* environments of the first case study, which underpins this conclusion. However, as we have seen by the experiments in the *AM-Env* environment of the second case study, this can be a wrong conclusion, because outdated pheromones may cause a lot of confusion. Especially when only a few routes are available and the vehicles approach the pickup stations nearly by the same routes, an individual configuration of different types of infochemicals (as in the *SAPi* series) achieves a much better solution efficiency. Particularly, when the communication costs cannot be neglected, which is usually the case, the best solution efficiency and solution process efficiency can be achieved with individually configured infochemicals.

However, the experimental results regarding the IBC approach have also revealed some of the runtime insufficiencies of self-organizing emergent MASs (see Section 5.1). The reactiveness, the greediness, the absence of global knowledge, as well as the inability to 'look into the future' of the agents result in several inefficiencies (see for instance the results on the number of blocking agents). But the experimental results and their analysis regarding the EIA approach have demonstrated that this approach is actually able to improve the efficiency of such self-organizing emergent systems at runtime. We have evaluated the capabilities of the approach in crafted scenarios (in which the layout of the stations and the order of the tasks were intended to provoke a situation in which the solution of the basic system coordinating by means of PIC is known to be suboptimal), random scenarios (which have not been devised to explicitly provoke bad behavior), and changing scenarios (in which the set of recurring tasks changes over time). The results have shown that the EIA approach is able to enhance the performance up to 28% of the basic solution, even in those scenarios, which were created randomly. Thus, the solutions of the basic system can also be improved under circumstances where such optimizations are not obvious for human observers. The adaptability of the approach can be observed in the results for the scenarios with a changing task set.

Despite these good results of the IBC approach and the EIA approach, admittedly, both approaches have their limitations and hence can be further improved in future. Thus, in the next chapter, we will summarize all conclusions, discuss these limitations, as well as provide an outlook on future work.

Chapter 9

Conclusions and Outlook

In this thesis we have developed various models, mechanisms, and architectures that simplify the design of efficient self-organizing emergent systems as well as facilitate their efficient operation. Section 9.1 summarizes these results and emphasizes their contributions with regard to the problems and challenges introduced at the beginning of this thesis. Section 9.2 critically discusses certain characteristics and limitations of the contributions as well as comments on certain decisions. Based on this discussion, Section 9.3 lists different aspects, how to extend and improve the developed approaches by future work.

9.1 Summary

The starting situation of this thesis was that self-organizing emergent systems are required by many companies in order to enable agile, flexible, scalable, robust, and adaptive computer systems, which generate only minimal operational expenditures due to their high degree of autonomy. However, several challenging problems prevent these systems from being practically applicable for the effective and efficient usage in industrial settings. This thesis has tackled two of these problems.

First, the design of effective but yet efficient self-organizing emergent systems was too complex, time-consuming, and costly (see *Problem 1*). To tackle this problem, we have provided a versatile and coherent model that allows for an efficient decentralized coordination of system elements (see *Challenge 1*). Therefore, we have investigated the general principles behind IBC in biology and adopted them leading to the DIC model. The developed model enables an effective but efficient coordination of homogeneous and heterogeneous system elements by means of different types of digital infochemicals within one single coordination mechanism. Due to the expressiveness of the model and the plethora of inspiring examples yet not adopted, we were able to specify an efficient DCM inspired by the pollination of flowers by honey bees, which fulfills the solution requirements of PDPs (see *Challenge 2*). Moreover, we have provided a design pattern, which allows for a time-saving and cost-effective, systematic engineering of self-organizing solutions grounded on IBC, design guidelines, which support engineers in identifying, specifying, and adapting new DCMs grounded on IBC, as well as a simulation tool, which supports engineers in identifying and selecting the most suitable and efficient coordination model(s) and mechanism(s) according to the system requirements (see *Challenge 3*). All of

these developed artifacts simplify the design of more efficient self-organizing emergent systems and reduce development time and costs. The conducted experiments regarding the IBC approach have proven the increased efficiency of these systems.

Second, due to several runtime insufficiencies the efficiency of self-organizing emergent systems during their operation could not be guaranteed in all situations (see *Problem 2*). Therefore, we have developed the EIA approach, which implements the principles of self-adaptation in order to compensate for these runtime insufficiencies. The EIA approach considers in particular the general constraints for the adaptation of self-organizing emergent systems, which result from their specific system characteristics compared to conventional computer systems. That is, the EIA approach takes into account the low observability and poor controllability of self-organizing emergent systems (see *Challenge 4*). This facilitates the adaptation of self-organizing emergent systems in a wider range of application domains, in which communication may be very costly, locally forbidden, globally restricted, structurally infeasible, or only temporally possible. The individual adaptation of the local behavior of system elements is accomplished by providing the elements with advice in the form of exception rules, how to behave more optimally in predicted future situations. All advices preserve the basic self-organizing and emergent behavior of the system elements as well as the resulting beneficial properties such as scalability, robustness, flexibility, and adaptivity (see *Challenge 5*). That is, all problem-solving decisions are still being made by the system elements themselves. Therefore, advice can be ignored by the elements and in particular will be ignored, if it has not been applied for a certain amount of time, e. g. if a predicted situation did not occur from some time on any more. This is a tribute to the openness and autonomy of self-organizing emergent systems (see *Challenge 6*). The conducted experiments regarding the EIA approach have proven the achieved improvement of the efficiency of these systems.

To demonstrate the capabilities of the developed concepts and the perfect complement to each other, we have instantiated both approaches for the efficient self-organizing emergent but at the same time also self-adaptive solution to dynamic PDPs. As a result, we have achieved a balanced mixture of the advantages of decentralized and centralized solution methods, i. e. a more flexible and faster solution compared to the one constructed by a fully centralized method but also a more efficient solution compared to the one constructed by a fully decentralized method. The experimental evaluation has underpinned this achievement as well.

9.2 Discussion

Despite having moved forward substantially in solving the two key problems around efficient self-organizing emergent systems, in this section we fairly discuss the developed approaches and concepts. For better readability, we separate this discussion up into two parts according to the respective approach.

IBC Approach

Without any doubt, the artifacts developed in this thesis with regard to the IBC approach simplify the design of efficient self-organizing emergent systems. However, one has to be aware of the fact that designing these systems in general remains more complicated than designing conventional computer systems, as the vague coherence between the microscopic and the macroscopic level of these systems in general still exists. But if the context and the solution requirements of a problem in hand align with the functionality provided by the DIC model (which can be checked by the developed design pattern), the artifacts developed in this thesis will reduce this 'micro-macro gap' to some extent and consequently will reduce the design complexity, save development time and development costs. Even if we can prove this reduced complexity qualitatively by the verifiable efficiency results of the designed systems (see Subsection 8.3.3), the effectiveness of existing engineering methodologies (cf. Section 3.4) does not allow us to prove this achievement quantitatively, yet. Moreover, some other aspects of the IBC approach have to be discussed as well:

- **Applicability:** As indicated, the IBC approach provides by no means an all-in-one coordination model suitable for all purposes and all domains. It rather is suitable for solutions that require e. g. the allocation of resources, group formation, information dissemination, indirect interaction, or spatial shapes and distributions, while providing a high degree of flexibility, robustness, scalability, adaptivity, and autonomy. To identify if such a decentralized solution is applicable to a problem in hand, an engineer has to consider the context, problem, forces, and solution sections of the developed design pattern (see Section 4.3). Please note that despite the discussed limitations of centralized solutions, for certain problems they still remain favorable compared to decentralized solutions, e. g. due to their higher observability and better controllability.

- **Design guidelines:** If the IBC approach has been identified as being suitable for solving a complex dynamic problem, the design guidelines developed in this thesis provide a good starting point for the identification, development, and adaptation of a new DCM grounded on IBC. Although the design guidelines therefore do not force an engineer to be a biological expert at the same time, they cannot entirely repudiate their biological origin. Thus, some basic knowledge in biology is still required to use the guidelines effectively.

- **Runtime inefficiencies:** One has to be careful not to construe the general runtime insufficiencies of environment-mediated MASs (see Subsection 7.3.2) as specific weaknesses of the DIC model or any instantiating DCM such as the PIC mechanism. In order to develop a decentralized coordination approach that compensates for these insufficiencies without any additional controller, one has to ensure that the agents (1) have enough processing power to compensate for their reactiveness and greediness on their own, (2) are able to share

all their knowledge immediately to gain global knowledge, and (3) are able to 'look into the future' to assign a dynamically appearing task to the best agent with respect to the global optimality of the solution. For a self-organizing emergent MAS, this constitutes quite a complex endeavor.

Apart from the inherent runtime insufficiencies, a few aspects of the PIC mechanism are worthwhile to discuss as well:

- **Scalability:** The more than 680.000 experiments regarding the PIC mechanism (see Section 8.3) were executed using several Windows Server 2008 R2 machines in parallel, each equipped with multiple Quad-Core Intel Xeon X5550 (2,67 GHz) processors. The execution time of a single PDP scenario on these machines took between less than a second for the smallest scenario with an efficient solution ($SAPi$ series) up to almost ten minutes for the largest scenario with an inefficient solution (S series). Thereby, the experimental results have proven that the PIC mechanism scales not only with the number of participating agents, but also with the number of tasks to fulfill and the size of the environment. Measuring the computation time within a single agent or location of the environment with an increasing amount of information to process, however, would have required some hundreds of independent processors and machines, which were not available.

- **Deadlocks and starvation:** For environments with directed connections between two locations, the evasion mechanism (see Algorithm 7.9) integrated in the PIC mechanism usually prevents the vehicle agents from falling into a deadlock or starvation. By contrast, for environments with undirected connections, similar to any other coordination mechanism, deadlocks cannot be excluded. However, these environments are very unlikely for the considered application domain.

- **Fairness:** Because fairness contradicts the inherent flexibility of self-organizing emergent solutions, the current specification of the PIC mechanism does not guarantee fairness, i. e. a task that appears prior to another task not necessarily will be served in the same order. Moreover, the specification even does not guarantee weak fairness, i. e. an appeared task may not be served ever.

- **Benchmarking:** The insufficiencies of existing centralized (OR) solution methods (see Subsection 7.1.3.1) for very dynamic and complex PDPs, which we were interested in in this thesis, do not allow for the calculation of optimal solutions to these problems. As a consequence, this fact does not allow for a comparison between the performance of a decentralized solution produced by the PIC mechanism and an optimal solution. Moreover, as commonly agreed benchmarking sets do not exist as well, a comparison to many existing decentralized (see Subsection 7.1.3.2) or hybrid (see Subsection 7.1.3.3) solution methods is not possible either.

EIA Approach

Some aspects of the EIA approach have to be discussed as well:

- **Applicability:** Any instantiation of the EIA approach requires three premises to be fulfilled by the controlled system (see Section 6.2). That is, each agent of the controlled system must be able (1) to collect and dump its local history, and (2) to incorporate exception rules in its local decision making. Usually, an agent is not required to reason about these rules. As a consequence, these two premises do not require an extensive amount of resources for data storage or computation and hence can be fulfilled by most applications. Furthermore, the EIA approach thus can be applied to a wide range of DCMs (see e. g. Section 3.3). The third premise, that is a sequence of run instances must have a (sub)set of similar tasks in (nearly) each run instance of the sequence, is usually fulfilled by most problems in everyday life as well. For all other problems, in which consequently no prediction is possible, no adaptation approach will be successful and the agents of the system e. g. really would have to be able to 'look into the future'.
- **Scalability:** The scalability of any EIA instantiation depends significantly on the capabilities of the optimization algorithm used within the instantiation to optimize the solution to the set of recurring tasks. However, as already mentioned, adequate OR solution methods for very complex problems, even static ones, are very rare and usually not publicly available (cf. Section 7.1.3). In the EIA instantiation presented in this thesis for the solution to PDPs, we thus have implemented our own branch-and-bound algorithm for a proof of concept. As a consequence, the set of recurring tasks as well as the number of agents that could be adapted by this instantiation was limited. Even if it can be expected that the EIA approach scales well for larger problems with a larger set of recurring tasks, this claim cannot be verified experimentally at the moment. Nonetheless, the question remains (even in the self-adaptation community), if the scalability in terms of thousands or millions of agents will be an issue for real-world application domains, or can we assume to split such big problems into several hierarchical subproblems only requiring some hundreds of agents to participate at most?
- **Efficiency guarantees:** Although an instantiation of the EIA approach is able to improve the efficiency of self-organizing emergent systems at runtime, strict guarantees on the efficiency that can be achieved or maintained by the advised system cannot be provided, yet. For instance, due to the generally low observability of these systems, the efficiency might decrease before the EIA was ever able to learn about the problem and based on that to adapt the system. Furthermore, even though an adaptation might improve the solution efficiency of the advised system in certain situations, it cannot be guaranteed, yet, that for other, possibly unexpected situations these exception rules are counterproductive and thus will even worsen the solution.

9.3 Outlook

Based on the summary and discussion of the IBC and the EIA approach in the last section, several aspects to investigate in future are conceivable for each approach. Please note that the aspects mentioned in this section refer largely to the general approaches, whereas the aspects mentioned in Section 7.4 refer to the corresponding application domain specific instantiations.

IBC Approach

The experimental results and their analysis regarding the IBC approach have demonstrated that the decentralized coordination by means of multiple digital infochemicals with different functions, dynamics, and semantics allows for the design of more efficient self-organizing emergent systems. Despite these substantial achievements, a few aspects remain for future work:

- **Identification of further DCMs grounded on IBC:** Because IBC is the most universally employed communication and coordination model between organisms in biology, it provides a plethora of inspiring examples that have not been adopted as DCMs for self-organizing emergent solutions yet. Apart from the pheromone-based coordination and PIC mechanism, thus, the IBC approach developed in this thesis has laid the foundation for the investigation of a new source of inspiration and consequently the DIC model serves for a variety of future instantiations also for other application domains apart from PDP. One example not further explained in this thesis is the *indirect defense strategy* applied by a specific species of plants [DvPdB03]. In order to get rid of herbivores, these plants emit certain scents (synomones) that attract natural enemies of these herbivores, which upon their arrival deter or destroy the attacking herbivores but not the plants. This inspiration has e. g. been used for the realization of a self-organizing emergent MAS based on the DIC model for the real-time control of water distribution systems [DDKB10].

- **Extension of design guidelines:** Because the design guidelines cannot entirely repudiate their biological origin, they may be enhanced in future. In particular, the biological knowledge required to develop new DCMs should be reduced to a minimum. Furthermore, the guidelines in future should provide a more detailed step-by-step support, how to identify a new DCM based on the solution requirements. This in turn requires the integration of the guidelines into an engineering methodology. However, today it is not yet clear which methodology is most appropriate for the engineering of self-organizing emergent systems in general (cf. Section 3.4).

- **Continuous parameter optimization:** Because environment-mediated coordination mechanisms in general are quite parameter sensitive, the proper

tuning of parameters has significant impact on the solution quality. However, the optimal configuration of the parameters depends significantly on the number and the distribution of the tasks to fulfill, the number of vehicles participating in the solution, and the size of the environment (as shown by the experimental analysis in Subsection 8.3.4). Thus, a kind of continuous optimization not only of the system behavior but also of certain parameters regarding the used DCM is required. It has to be evaluated, if this optimization can be handled by the EIA as well, or if additional control approaches are required.

EIA Approach

The experimental results and their analysis regarding the EIA approach have shown that the concept of an EIA as well as its instantiation for PDPs is powerful enough to improve the efficiency of self-organizing emergent systems at runtime. Nevertheless, some aspects remain for future work as well:

- **More sophisticated optimization algorithms:** As the scalability of any EIA instantiation significantly depends on the applied optimization algorithm, the application and integration of more sophisticated optimization algorithm will consequently increase the scalability of an EIA instantiation as well as its provided solution quality. Because in many cases the exact optimal solution is not always required, algorithms that do not claim to generate the optimal solution might come up with a sufficient solution much faster than e. g. the branch-and-bound algorithm applied in this thesis. Because the application of an optimization algorithm also depends on the problem domain, the use of meta-heuristics such as genetic algorithms, which can be applied to arbitrary application domains, is recommended.

- **Conflict resolution mechanism:** Since the definition of a rule-applying agent includes a conflict resolution mechanism that decides which exception rule(s) to apply in case that more than one exception rule is applicable, a similar concept has to be included in an EIA. Because in the presented instantiation an EIA derives only one type of exception rules (ignore rules), this was not necessary so far. However, as soon as more types of exception rules can be derived by an EIA, the latter has to decide, which type(s) of exception rule(s) to derive for the adaptation of the controlled system. So, it has to be investigated, if existing conflict resolution mechanisms from the rule community are applicable for the EIA requirements.

- **Decentralized/hierarchical EIA architectures:** In the current specification, an EIA is designed as an independent agent that runs centrally along with the agents of the advised system. However, other architectural alternatives are conceivable as well. For instance, every system element can be attributed with

its own EIA unit, while these units similarly then have to coordinate their actions and interactions to achieve a certain functionality. Likewise, these local EIA units could also be advised by an additional central EIA unit, which then represents a hierarchical architecture. For both architectural alternatives it has to be proven, however, in which application domains they are more appropriate, which additional functionality can be provided to the advised system, and which communication overhead such architectures generate!

- **Design guidelines for an EIA:** In the same way as the design guidelines for the IBC approach help to instantiate the DIC model, design guidelines for the EIA approach will be required in order to instantiate the abstract functions of the EIA model. In particular for new problem domains, engineers for instance have to be supported in answering questions like: which local information of agents has to be collected by an EIA, how to identify recurring tasks in the global history, and how to identify exception rules that will improve the efficiency of the controlled system?

- **Handle uncertainty and noise:** In various application domains, information does not necessarily need to be reliable. When the information the agents collected is outdated (e. g. because the environment changed in the meantime) or the sensors can not guarantee 100% accuracy, the EIA will have to deal with these limitations and work around them, e. g. by applying a probability factor to every piece of information. Storing such data and working with it might be very different from the way it is done now and might require new solutions that were not thought of yet.

- **Simulation of derived exception rules:** For certain application domains, the reaction of the agents to a new set of exception rules must not necessarily be simulated before the exception rules are sent to the agents. In such cases, the next run instance can be used to evaluate and learn, if the exception rules have helped the system to perform better and to derive further optimizations. Due to the possible diversity of exception rules, for certain application domains, however, an EIA has to be capable of evaluating the effects of an exception rule (set) first, to identify the most appropriate rule (set) to apply in a specific situation, before deploying the rule (set) to the basic agents. This allows an EIA to test in advance, how the agents will behave to a new exception rule (set) before it is communicated to the agents. This kind of self-reflection is usually achieved with simulations on a model of the system [Ste05]. If the simulation result is still too bad, the rule creation step can be repeated until the simulated solution is satisfactory. Only then all derived exception rules will be sent to the real agents. Another option works similar to the function of dreams in humans. [BIRB04] defines a "dreaming" state for applications in which operations can be executed and evaluated on the system itself but data modified by the actions is not persisted. This resembles a simulation on the real system with current, real data. However, one has to be aware of the fact

that such simulations can be computationally prohibitive due to an almost infinite state space and may require quite a lot of time.

- **Incorporation of risk analysis:** In order to provide certain efficiency guarantees, it is not only useful to know, which degree of efficiency can be achieved due to an adaptation of the system, but also, how worse a solution might become in unexpected situations in particular due to the adaptation. This requires an online test system that tests based on the last known state of the advised system and of the EIA in an integrated simulation environment the adaptations by the EIA to certain provoked situations as well as the corresponding reaction of the adapted system. Based on the results the test system can identify possibly awkward situations in advance. As a consequence, it can, for instance, 'advice' the EIA not to perform certain adaptations, when such an awkward situation is detected in the real system. Such a test system could even more advice the EIA to reverse certain prior adaptations of the system, i. e. to adapt or delete certain exception rules already communicated to the agents, as these rules may potentially worsen the global solution in these awkward situations. Furthermore, the test system could inform the user of the entire system, which potential risk in terms of worsening the solution exists for the system at any point of time. On the other side, this potential risk could also be utilized by the EIA in a conflict resolution mechanism for the derivation of different types of exception rules. That is, the EIA sends exactly this type of exception rule to the agents, which exhibits the lowest risk.

Even though (advised) self-organizing emergent solutions will hardly achieve the efficiency of centralized solutions, their acceptance and application in particular in industrial settings will become more profitable in future, if the design of efficient solutions becomes easier and the benefits of adaptivity, flexibility, robustness, and scalability compensate during operation for certain efficiency drawbacks. The contributions of this thesis represent a major step towards this objective.

Bibliography

[AAC+00] Harold Abelson, Don Allen, Daniel Coore, Chris Hanson, George Homsy, Thomas F. Knight Jr., Radhika Nagpal, Erik Rauch, Gerald Jay Sussman, and Ron Weiss. Amorphous Computing. *Communications of the ACM*, 43(5):74–82, May 2000.

[Abb06] Russ Abbott. Emergence explained: Abstractions: Getting epiphenomena to do real work. *Complexity*, 12(1):13–26, 2006.

[ABI07] Richard Anthony, Alan Butler, and Mohammed Ibrahim. Exploiting Emergence in Autonomic Systems. In Manish Parashar and Salim Hariri, editors, *Autonomic Computing: Concepts, Infrastructure, and Applications*, pages 121–148. CRC Press, 2007.

[ACMS06] Sandip Agarwala, Yuan Chent, Dejan Milojicic, and Karsten Schwan. QMON: QoS- and utility-aware monitoring in enterprise systems. In *Proceedings of the 3rd IEEE International Conference on Autonomic Computing (ICAC 2006)*, pages 124–133. IEEE Computer Society, 2006.

[ACP93] Jean-Marc Andreoli, Paolo Ciancarini, and Remo Pareschi. Interaction abstract machines. In *Research Directions in Concurrent Object Oriented Programming*, pages 257–280. MIT Press, 1993.

[ADR03] Dave Anderson, Jim Dykes, and Erik Riedel. More Than an Interface — SCSI vs. ATA. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST 2003)*, pages 245–257. USENIX Association, 2003.

[Age08] Agentsheets. `http://www.agentsheets.com`, 2008.

[Agh86] Gul Agha. *Actors: a model of concurrent computation in distributed systems*. MIT Press, Cambridge, MA, USA, 1986.

[AGMS87] Jeffrey C. Alexander, Bernhard Giesen, Richard Münch, and Neil J. Smelser, editors. *The Micro-Macro Link*. University of California Press, 1987.

[Ago92] William C. Agosta. *Chemical communication: the language of pheromones*. Scientific American Library, New York, 1992.

[AHS93] Farhad Arbab, Ivan Herman, and Per Spilling. An overview of Manifold and its implementation. *Concurrency: Practice and Experience*, 5(1):23–70, 1993.

[AK07] Sherif Abdelwahed and Nagarajan Kandasamy. A Control-Based Approach to Autonomic Performance Management in Computing Systems. In Manish Parashar and Salim Hariri, editors, *Autonomic Computing: Concepts, Infrastructure, and Applications*, pages 149–167. CRC Press, 2007.

[AKOS09] Asama, Kurokawa, Ota, and Sekiyama, editors. *Distributed Autonomous Robotic Systems*, volume 8. Springer, 2009.

[AL98] Yariv Aridor and Danny B. Lange. Agent design patterns: elements of agent application design. In *Proceedings of the 2nd International Conference on Autonomous Agents (AGENTS 1998)*, pages 108–115. ACM, 1998.

[Ale77] Christopher Alexander. *A Pattern Language. Towns, Buildings, Construction.* Oxford University Press, New York, 1977.

[Ale90] Samuel Alexander. *Space, Time, and Deity: Gifford Lectures, 1916-1918.* Thoemmes Continuum, 1990.

[Ame03] America Power Conversion. Determining Total Cost of Ownership for Data Center and Network Room Infrastructure. http://www.apcmedia.com/salestools/CMRP-5T9PQG_R2_EN.pdf, December 2003.

[And72] Philip W. Anderson. More is Different. *Science*, 177(4047):393–396, 1972.

[And02] Carl Anderson. Self-Organization in Relation to Several Similar Concepts: Are the Boundaries to Self-Organization Indistinct? *The Biological Bulletin*, 202(3):247–255, 2002.

[And04] Carl Anderson. Linking Micro- to Macro-level Behavior in the Aggressor-Defender-Stalker Game. *Adaptive Behavior - Animals, Animats, Software Agents, Robots, Adaptive Systems*, 12(3-4):175–185, 2004.

[Ant08] Richard Anthony. Scalable and Efficient Graph Colouring in 3 Dimensions using Emergence Engineering Principles. In Sven Brueckner, Paul Robertson, and Umesh Bellur, editors, *Proceedings of the 2nd IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2008)*, pages 370–379. IEEE Computer Society, 2008.

[Any08] AnyLogic. http://www.xjtek.com, 2008.

[AP91] Jean-Marc Andreoli and Remo Pareschi. Linear Ojects: Logical Processes with Built-in Inheritance. *New Generation Computing*, 9(3/4):445–474, 1991.

[APS04] Samir Aknine, Suzanne Pinson, and Melvin F. Shakun. An Extended Multi-Agent Negotiation Protocol. *Autonomous Agents and Multi-Agent Systems*, 8(1):5–45, 2004.

[Arb96] Farhad Arbab. The IWIM Model for Coordination of Concurrent Activities. In *Proceedings of the 1st International Conference on Coordination Languages and Models (COORDINATION 1996)*, volume 1061 of *Lecture Notes in Computer Science*, pages 34–56. Springer, 1996.

[Arb98] Farhad Arbab. What Do You Mean, Coordination? In *Bulletin of the Dutch Association for Theoretical Computer Science (NVTI)*, 1998.

[Arb04] Farhad Arbab. Reo: a channel-based coordination model for component composition. *Mathematical Structures in Computer Science*, 14(3):329–366, 2004.

[Ari24] Aristotle. *Metaphysics, Book VIII*. Clarendon Press, Oxford, 1924. translated by W. D. Ross.

[ASB02] Tarek F. Abdelzaher, Kang G. Shin, and Nina Bhatti. Performance Guarantees for Web Server End-Systems: A Control-Theoretical Approach. *IEEE Transactions on Parallel and Distributed Systems*, 13(1):80–96, 2002.

[Asc08] Ascape. http://ascape.sourceforge.net, 2008.

[Ash47] William Ross Ashby. Principles of the Self-Organizing Dynamic System. *Journal of General Psychology*, 37:125–128, 1947.

[Ash60] William Ross Ashby. *Design for a Brain*. Chapman & Hall, London, 1960.

[Ash62] William Ross Ashby. Principles of Self-Organization. In Heinz von Foerster and George W. Zopf Jr., editors, *Principles of Self-Organization: Transactions of the University of Illinois Symposium on Self-Organization*, pages 255–278, New York, 1962. Pergamon Press.

[Aut04] Autonomic Communication. http://www.autonomic-communication.org, 2004.

[AW94] Karl J. Astrom and Björn Wittenmark. *Adaptive Control*. Prentice Hall, 2nd edition, 1994.

[Ban96] Mario Banville. Sonia: An Adaptation of LINDA for Coordination of Activities in Organisations. In *Proceedings of the 1st International Conference on Coordination Languages and Models (COORDINATION 1996)*, volume 1061 of *Lecture Notes in Computer Science*, pages 57–74. Springer, 1996.

[BAU+10] Birger Becker, Florian Allerding, Reiner Ulrich, Mattias Kahl, Urban Richter, Daniel Pathmaperuma, Hartmut Schmeck, and Thomas Leibfried. Decentralized Energy-Management to Control Smart-Home Architectures. In *Proceedings of the 23rd International Conference on Architecture of Computing Systems (ARCS 2010)*, volume 5974 of *Lecture Notes in Computer Science*, pages 150–161. Springer, 2010.

[BB97] Mathieu Buffo and Didier Buchs. A Coordination Model for Distributed Object Systems. In *Proceedings of the 2nd International Conference on Coordination Languages and Models (COORDINATION 1997)*, volume 1282 of *Lecture Notes in Computer Science*, pages 410–413. Springer, 1997.

[BBB04] Jenna Burrell, Tim Brooke, and Richard Beckwith. Vineyard computing: Sensor networks in agricultural production. *IEEE Pervasive Computing*, 3(1):38–45, 2004.

[BBF+08] Michael Beigl, Uwe Brinkschulte, Fridtjof Feldbusch, Dietmar Fey, Stefan Fischer, Christian Hochberger, Rolf Hoffmann, Wolfgang Karl, Jochen Krebs, Jürgen Kleinöder, Klaus Lagally, Falk Langhammer, Paul Lukowicz, Peter Marwedel, Erik Maehle, Christian Müller-Schloer, Burghardt Schallenberger, Hartmut Schmeck, Djamshid Tavangarian, Wolfgang Trumler, Theo Ungerer, and Klaus Waldschmidt. Grand Challenges der Technischen Informatik. Report, Gesellschaft für Informatik e.V., March 2008.

[BC91] Antonio Brogi and Paolo Ciancarini. The concurrent language, Shared Prolog. *ACM Transactions on Programming Languages and Systems*, 13(1):99–123, 1991.

[BC00] Mark H. Bickhard and Donald T. Campbell. Emergence. In Peter Bøgh Andersen, Claus Emmeche, Niels Ole Finnemann, and Peder Voetmann Christiansen, editors, *Downward Causation*. Aarhus University Press, Aarhus, 2000.

[BC02] Piermarco Burrafato and Massimo Cossentino. Designing a multi-agent solution for a bookstore with the PASSI methodology. In

Proceedings of the 4th International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS 2002), volume 57 of *CEUR Workshop Proceedings*, 2002.

[BC03] Laura Bocchi and Paolo Ciancarini. A Perspective on Multiagent Coordination Models. In *Communications in Multiagent Systems*, volume 2650 of *Lecture Notes in Computer Science*, pages 146–163. Springer, 2003.

[BCD+06] Özalp Babaoglu, Geoffrey Canright, Andreas Deutsch, Gianni A. Di Caro, Frederick Ducatelle, Luca M. Gambardella, Niloy Ganguly, Márk Jelasity, Roberto Montemanni, Alberto Montresor, and Tore Urnes. Design patterns from biology for distributed computing. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 1(1):26–66, September 2006.

[BCGL07] Gerardo Berbeglia, Jean-François Cordeau, Irina Gribkovskaia, and Gilbert Laporte. Static Pickup and Delivery Problems: A Classification Scheme and Survey. *TOP*, 15(1):1–31, 2007.

[BCL10] Gerardo Berbeglia, Jean-François Cordeau, and Gilbert Laporte. Dynamic pickup and delivery problems. *European Journal of Operational Research*, 202(1):8–15, 2010.

[BDG+09] Yuriy Brun, Giovanna Di 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 Betty H. C. Cheng, Rogerio de Lemos, Holger Giese, Paola Inverardi, and Jeff Magee, editors, *Software Engineering for Self-Adaptive Systems*, volume 5525 of *Lecture Notes in Computer Science*, pages 48–70. Springer, 2009.

[BDT99] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, 1999.

[Bed97] Mark A. Bedau. Weak Emergence. In James E. Tomberlin, editor, *Philosophical Perspectives: Mind, Causation, and World*, volume 11, pages 375–399. Blackwell Publishers, Malden, MA, USA, 1997.

[Bed02] Mark A. Bedau. Downward causation and the autonomy of weak emergence. *Principia*, 6:5–50, 2002.

[Bee95] Randall D. Beer. A dynamical systems perspective on agent-environment interaction. *Artificial Intelligence*, 72(1-2):173–215, 1995.

[BEW70] William L. Brown, Jr., Thomas Eisner, and Robert H. Whittaker. Allomones and kairomones: Transpecific chemical messengers. *BioScience*, 20:21–22, 1970.

[BF03] David Bantz and David Frank. Challenges in autonomic personal computing with some new results in automatic configuration management. In *Proceedings of the IEEE International Conference on Industrial Informatics (INDIN 2003)*, pages 451–456. IEEE Computer Science, 2003.

[BFM01] Jean-Pierre Banâtre, Pascal Fradet, and Daniel Le Métayer. Gamma and the Chemical Reaction Model: Fifteen Years After. In *Proceedings of the Workshop on Multiset Processing (WMP 2000)*, volume 2235 of *Lecture Notes In Computer Science*, pages 17–44. Springer, 2001.

[BFV00] Hans-Jürgen Bürckert, Klaus Fischer, and Gero Vierke. Holonic Transport Scheduling with Teletruck. *Applied Artificial Intelligence*, 14(7):697–725, 2000.

[BGP06] Carole Bernon, Marie Pierre Gleizes, and Gauthier Picard. Enhancing Self-organising Emergent Systems Design with Simulation. In *Proceedings of the 7th International Workshop on Engineering Societies in the Agents World (ESAW 2006), Revised Selected and Invited Papers*, pages 284–299, 2006.

[BGPP02] Carole Bernon, Marie-Pierre Gleizes, Sylvain Peyruqueou, and Gauthier Picard. ADELFE: A Methodology for Adaptive Multi-agent Systems Engineering. In *Proceedings of the 3rd International Workshop on Engineering Societies in the Agents World (ESAW 2002), Revised Papers*, volume 2577 of *Lecture Notes in Computer Science*, pages 70–81. Springer, 2002.

[BH06] Russell Bent and Pascal Van Hentenryck. A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. *Computers and Operations Research*, 33(4):875–893, 2006.

[BHJY07] Sven Brückner, Salima Hassas, Márk Jelasity, and Daniel Yamins, editors. *Proceedings of the 4th International Workshop on Engineering Self-Organising Systems (ESOA 2006)*, volume 4335 of *Lecture Notes in Computer Science*. Springer, 2007.

[BHM96] Achim Bachem, Winfried Hochstättler, and Martin Malich. The Simulated Trading Heuristic for Solving Vehicle Routing Problems. *Discrete Applied Mathematics*, 65(1–3):47–72, 1996.

[BHRS98] Joachim Baumann, Fritz Hohl, Kurt Rothermel, and Markus Straßer. Mole – Concepts of a mobile agent system. *World Wide Web*, 1(3):123–137, 1998.

[BIRB04] Alun R. Butler, Mohamed Ibrahim, Keith Rennolls, and Liz Bacon. On The Persistence of Computer Dreams - An Application Framework for Robust Adaptive Deployment. In *Proceedings of the Database and Expert Systems Applications (DEXA 2004)*, pages 716–720. IEEE Computer Society, 2004.

[BJ08] Özalp Babaoglu and Márk Jelasity. Self-* Properties Through Gossiping. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 366(1881):3747–3757, 2008.

[BJ09] Dariusz Barbucha and Piotr Jedrzejowicz. Agent-Based Approach to the Dynamic Vehicle Routing Problem. In *Proceedings of the 7th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS 2009)*, volume 55 of *Advances in Soft Computing*, pages 169–178. Springer, 2009.

[BJM04] Özalp Babaoglu, Márk Jelasity, and Alberto Montresor. Grassroots Approach to Self-Management in Large-Scale Distributed Systems. In *Proceedings of the EU-NSF Strategic Research Workshop on Unconventional Programming Paradigms (UPP 2004)*, volume 3566 of *Lecture Notes in Computer Science*, pages 286–296. Springer, 2004.

[BJM+05] Özalp Babaoglu, Márk Jelasity, Alberto Montresor, Christof Fetzer, Stefano Leonardi Aad van Moorsel, and Maarten van Steen, editors. *Self-star Properties in Complex Information Systems*, volume 3460 of *Lecture Notes in Computer Science*. Springer, 2005.

[BM90] Jean-Pierre Banâtre and Daniel Le Métayer. The GAMMA Model and Its Discipline of Programming. *Science of Computer Programming*, 15(1):55–77, 1990.

[BM93] Stefan Bussmann and Jörg P. Müller. A Negotiation Framework for Cooperating Agents. In *Proceedings of the Special Interest Group on Cooperating Knowledge Based Systems (CKBS-SIG 1992)*, pages 1–17, University of Keele, 1993. DAKE Centre.

[BML+05] Mark Brodie, Sheng Ma, Guy Lohman, Laurent Mignet, Natwar Modani, Mark Wilding, Jon Champlin, and Peter Sohn. Quickly finding known software problems via automated symptom matching. In *Proceedings of the 2nd International Conference on Automatic Computing (ICAC 2005)*, pages 101–110. IEEE Computer Society, 2005.

[BMM02] Özalp Babaoglu, Hein Meling, and Alberto Montresor. Anthill: A Framework for the Development of Agent-Based Peer-to-Peer Systems. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS 2002)*, page 15. IEEE Computer Society, 2002.

[BMMS+06] Jürgen Branke, Moez Mnif, Christian Müller-Schloer, Holger Prothmann, Urban Richter, Fabian Rochner, and Hartmut Schmeck. Organic Computing - Addressing Complexity by Controlled Self-organization. In Tiziana Margaria, Anna Philippou, and Bernhard Steffen, editors, *Proceedings of ISoLA 2006*, pages 200–206, Paphos, Cyprus, November 2006.

[BMMZ03] Nadia Busi, Cristian Manfredini, Alberto Montresor, and Gianluigi Zavattaro. PeerSpaces: data-driven coordination in peer-to-peer networks. In *Proceedings of the 2003 ACM Symposium on Applied Computing (SAC 2003)*, pages 380–386. ACM, 2003.

[BMND05] Jürgen Branke, Martin Middendorf, Guntram Noeth, and Maged Dessouky. Waiting Strategies for Dynamic Vehicle Routing. *Transportation Science*, 39(3):298–312, 2005.

[BMO01] Bernhard Bauer, Jörg P. Müller, and James Odell. Agent UML: A Formalism for Specifying Multiagent Software Systems. In *1st International Workshop on Agent-Oriented Software Engineering (AOSE 2000), Revised Papers*, volume 1957 of *Lecture Notes in Computer Science*, pages 109–120. Springer, 2001.

[BN97] Stephen L. Buchmann and Gary Paul Nabhan. *The forgotten pollinators*. Island Press, 1997.

[BNOT09] Cyrille Bertelle, Michel Nabaa, Damien Olivier, and Pierrick Tranouez. A Decentralised Approach for the Transportation On Demand Problem. In *From System Complexity to Emergent Properties*, volume 2009 of *Understanding Complex Systems*, pages 281–289. Springer, 2009.

[Bod40] Hendrik W. Bode. Relations Between Attenuation and Phase in Feedback Amplifier Design. *Bell Systems Technical Journal*, 19:421–454, 1940.

[Bon02] Eric Bonabeau. Predicting the Unpredictable. Harvard Business Review, March 2002.

[BP00] Sven Brückner and H. Van Dyke Parunak. Multiple Pheromones for Improved Guidance. In *Proceedings of the 2nd DARPA-JFACC Symposium on Advances in Enterprise Control*, 2000.

[BP05] Sven Brueckner and H. Van Dyke Parunak. Swarming Distributed Pattern Detection and Classification. In Danny Weyns, H. Van Dyke Parunak, and Fabien Michel, editors, *Proceedings of 1st International Workshop on Environments for Multi-Agent Systems (E4MAS 2004), Revised Selected Papers*, volume 3374 of *Lecture Notes in Computer Science*, pages 232–245. Springer, 2005.

[BPR01] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. JADE: a FIPA2000 compliant agent development environment. In *Proceedings of the 5th International Conference on Autonomous Agents (AGENTS 2001)*, pages 216–217. ACM, 2001.

[BRB08] Sven Brueckner, Paul Robertson, and Umesh Bellur, editors. *Proceedings of the 2nd IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2008)*. IEEE Computer Society, 2008.

[Bre01] Jonathan L. Bredin. *Market-based Control of Mobile-Agent Systems*. Phd thesis, Dartmouth College, 2001.

[Bre08] Breve. http://www.spiderland.org, 2008.

[Bro25] Charlie Dunbar Broad. *The Mind and its Place in Nature*. Kegan Paul, Trench, Trubner, 1925.

[Bro68] William L. Brown, Jr. An Hypothesis Concerning the Function of the Metapleural Glands in Ants. *The American Naturalist*, 102(924):188–191, 1968.

[Bro86] Rodney A. Brooks. A Robust Layered Control System For A Mobile Robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, 1986.

[Bro90] William L. Brogan. *Modern Control Theory*. Prentice Hall, 3rd edition, 1990.

[Brü00] Sven Brückner. *Return from the Ant - Synthetic Ecosystems for Manufacturing Control*. Phd thesis, Humboldt-Universität, Berlin, 2000.

[BSHZ06] Sven Brückner, Giovanna Di Marzo Serugendo, David Hales, and Franco Zambonelli, editors. *Proceedings of the 3rd International Workshop on Engineering Self-Organising Systems (ESOA 2005)*, volume 3910 of *Lecture Notes in Computer Science*. Springer, 2006.

[BSKN05] Sven Brückner, Giovanna Di Marzo Serugendo, Anthony Karageorgos, and Radhika Nagpal, editors. *Proceedings of the 2nd International Workshop on Engineering Self-Organising Systems, Methodologies and Applications (ESOA 2004)*, volume 3464 of *Lecture Notes in Computer Science*. Springer, 2005.

[BTD+97] Eric Bonabeau, Guy Theraulaz, Jean-Louis Deneubourg, Serge Aron, and Scott Camazine. Self-Organization in Social Insects. *Trends in Ecology and Evolution*, 12(5):188–193, 1997.

[Bur01] Roland Burns. *Advanced Control Engineering*. Butterworth-Heinemann, 2001.

[BW63] William H. Bossert and Edward O. Wilson. The analysis of olfactory communication among animals. *Journal of Theoretical Biology*, 5(3):443–469, 1963.

[BWD+93] Mario R. Barbacci, Charles B. Weinstock, Dennis L. Doubleday, Michael J. Gardner, and Randall W. Lichota. Durra: a structure description language for developing distributed applications. *IEEE Software Engineering Journal*, 8(2):83–94, 1993.

[BWHM04] Nelis Boucké, Danny Weyns, Tom Holvoet, and Koenraad Mertens. Decentralized Allocation of Tasks with Delayed Commencement. In G. Chiara, P. Ciorgini, and W. van der Hoek, editors, *European Workshop on Multi-Agent Systems (EUMAS 2004)*, pages 57–68, 2004.

[BZ01] Nadia Busi and Gianluigi Zavattaro. Publish/Subscribe vs. Shared Dataspace Coordination Infrastructures: Is It Just a Matter of Taste? In *Proceedings of the 10th IEEE International Workshops on Enabling Technologies (WETICE 2001)*, pages 328–333. IEEE Computer Society, 2001.

[Cam74] Donald T. Campbell. 'Downward causation' in hierarchically organised biological systems. In F.J. Ayala and T. Dobzhansky, editors, *Studies in the Philosophy of Biology: reduction and related problems*, pages 179–186. Macmillan Press, 1974.

[CAR05] Nuno Carvalho, Filipe Araujo, and Luis Rodrigues. Scalable QoS-Based Event Routing in Publish-Subscribe Systems. In *Proceedings of the 4th IEEE International Symposium on Network Computing and Applications (NCA 2005)*, pages 101–108. IEEE Computer Society, 2005.

[CB04] Eduardo F. Camacho and Carlos Bordons. *Model Predictive Control*. Springer, 2004.

[CBB+02] Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Robert Nord, and Judith Stafford. *Documenting Software Architectures: Views and Beyond*. SEI Series in Software Engineering. Addison-Wesley Professional, 2002.

[CC96] Stefania Castellani and Paolo Ciancarini. Enhancing Coordination and Modularity Mechanisms for a Language with Objects-as-Multisets. In *Proceedings of the 1st International Conference on Coordination Languages and Models (COORDINATION 1996)*, volume 1061 of *Lecture Notes in Computer Science*, pages 89–106. Springer, 1996.

[CCG+02] Giovanni Caire, Wim Coulier, Francisco Garijo, Jorge Gomez, Juan Pavón, Francisco Leal, Paulo Chainho, Paul Kearney, Jamie Stark, Richard Evans, and Philippe Massonet. Agent Oriented Analysis Using Message/UML. In *Proceedings of the 2nd International Workshop on Agent-Oriented Software Engineering (AOSE 2001), Revised Papers and Invited Contributions*, volume 2222, pages 119–135. Springer-Verlag, 2002.

[CCR96] Stefania Castellani, Paolo Ciancarini, and Davide Rossi. The ShaPE of ShaDe: a Coordination System. Technical Report UBLCS-96-5, University of Bologna, March 1996.

[CDF+01] Scott Camazine, Jean-Louis Deneubourg, Nigel R. Franks, James Sneyd, Guy Theraulaz, and Eric Bonabeau. *Self-Organization in Biological Systems*. Princeton University Press, 2001.

[CdF+06] Betty H. C. Cheng, Rogério de Lemos, Stephen Fickas, David Garlan, Jeff Magee, Hausi Müller, and Richard Taylor, editors. *Proceedings of the 1st International Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2006)*. ACM, 2006.

[CdF+07] Betty H. C. Cheng, Rogério de Lemos, Stephen Fickas, David Garlan, Marin Litoiu, Jeff Magee, Hausi Müller, and Richard Taylor, editors. *Proceedings of the 2nd International Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2007)*. IEEE Computer Society, 2007.

[CdG+08] Betty H. C. Cheng, Rogério de Lemos, David Garlan, Holger Giese, Marin Litoiu, Jeff Magee, Hausi Müller, and Richard Taylor, editors. *Proceedings of the 3rd International Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2008)*. ACM, 2008.

[CdG+09] Betty H. C. Cheng, Rogério de Lemos, David Garlan, Holger Giese, Marin Litoiu, Jeff Magee, Hausi Müller, and Richard Taylor, editors. *Proceedings of the 4th International Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2009)*. ACM, 2009.

[CDKR02] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony I. T. Rowstron. Scribe: a large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on on Selected Areas in Communications*, 20(8):1489–1499, 2002.

[CDSS02] Jean-François Cordeau, Jacques Desrosiers, Marius M. Solomon, and François Soumis. VRP with Time Windows. In Paolo Toth and Daniele Vigo, editors, *The Vehicle Routing Problem*, SIAM Monographs on Discrete Mathematics and Applications, pages 157–193. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2002.

[CEBÅ02] Anton Cervin, Johan Eker, Bo Bernhardsson, and Karl-Erik Årzén. Feedback-Feedforward Scheduling of Control Tasks. *Real-Time Systems*, 23(1–2):23–53, 2002.

[CG98] Luca Cardelli and Andrew D. Gordon. Mobile Ambients. In *Proceedings of the 1st International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 1998)*, volume 1378 of *Lecture Notes in Computer Science*, pages 140–155. Springer, 1998.

[CGGG03] Davy Capera, Jean-Pierre Georgé, Marie-Pierre Gleizes, and Pierre Glize. The AMAS theory for complex problem solving based on self-organizing cooperative agents. In *Proceedings of the 12th International Workshop on Enabling Technologies (WETICE 2003)*, pages 383–388. IEEE Computer Society, 2003.

[CGI+09] Betty H.C. Cheng, Holger Giese, Paola Inverardi, Jeff Magee, Rogerio de Lemos, Jesper Andersson, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, Giovanna Di Marzo Serugendo, Schahram Dustdar, Anthony Finkelstein, Cristina Gacek, Kurt Geihs, Vincenzo Grassi, Gabor Karsai, Holger Kienle, Jeff Kramer, Marin Litoiu, Sam Malek, Raffaela Mirandola, Hausi Müller, Sooyong Park, Mary Shaw, Matthias Tichy, Massimo Tivoli, Danny Weyns, and Jon Whittle. Software Engineering for Self-Adaptive Systems: A Research Road Map. In Betty H. C. Cheng, Rogerio de Lemos, Holger Giese, Paola Inverardi, and Jeff Magee, editors, *Software Engineering for Self-Adaptive Systems*, volume 5525 of *Lecture Notes in Computer Science*, pages 1–26. Springer, 2009.

[CGP99] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, 1999.

[CGS+02a] Shang-Wen Cheng, David Garlan, Bradley Schmerl, Jo ao Pedro Sousa, Bridget Spitznagel, Peter Steenkiste, and Ningning Hu.

Software Architecture-Based Adaptation for Pervasive Systems. In *Proceedings of the International Conference on Architecture of Computing Systems (ARCS 2002)*, volume 2299 of *Lecture Notes In Computer Science*, pages 67–82. Springer, 2002.

[CGS+02b] Shang-Wen Cheng, David Garlan, Bradley Schmerl, Peter Steenkiste, and Ningning Hu. Software Architecture-Based Adaptation for Grid Computing. In *Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing (HPDC 2002)*, page 389. IEEE Computer Society, 2002.

[CGS05] Shang-Wen Cheng, David Garlan, and Bradley Schmerl. Making Self-Adaptation an Engineering Reality. In *Proceedings of the International Workshop on Self-* Properties in Complex Information Systems (SELF-STAR 2004)*, volume 3460 of *Lecture Notes in Computer Science*, pages 158–173. Springer, 2005.

[CGZ95] Nicholas Carriero, David Gelernter, and Lenore D. Zuck. Bauhaus Linda. In *Proceedings of the Workshop on Models and Languages for Coordination of Parallelism and Distribution, Object-Based Models and Languages for Concurrent Systems (ECOOP 1994), Selected Papers*, volume 924 of *Lecture Notes in Computer Science*, pages 66–76. Springer, 1995.

[CHM+97] Barbara Chapman, Matthew Haines, Piyush Mehrotra, Hans Zima, and John Van Rosendale. Opus: A Coordination Language for Multidisciplinary Applications. *Scientific Programming*, 6(2):187–200, 1997.

[CHMS08] Emre Cakar, Jörg Hähner, and Christian Müller-Schloer. Creating Collaboration Patterns in Multi-agent Systems with Generic Observer/Controller Architectures. In *Proceedings of the 2nd International Conference on Autonomic Computing and Communication Systems (Autonomics 2008)*, pages 1–9. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[Cia91] Paolo Ciancarini. PoliS: a programming model for multiple tuple spaces. In *Proceedings of the 6th International Workshop on Software specification and design (IWSSD 1991)*, pages 44–51. IEEE Computer Society Press, 1991.

[Cia96] Paolo Ciancarini. Coordination models and languages as software integrators. *ACM Computing Surveys*, 28(2):300–302, 1996.

[CL03] Jean-François Cordeau and Gilbert Laporte. The Dial-a-Ride Problem (DARP): Variants, modeling issues and algorithms. *4OR*, 1(2):89–101, 2003.

[CL07] Jean-François Cordeau and Gilbert Laporte. The Dial-a-Ride Problem: Models and Algorithms. *Annals of Operations Research*, 153(1):29–46, 2007.

[Cla98] Andy Clark. *Being There: Putting Brain, Body, and World Together Again*. MIT Press, 1998.

[CLPS07] Jean-François Cordeau, Gilbert Laporte, Jean-Yves Potvin, and Martin W. P. Savelsbergh. Transportation on Demand. In Cynthia Barnhart and Gilbert Laporte, editors, *Handbooks in Operations Research and Management Science: Transportation*, volume 14, pages 429–466. Elsevier, Amsterdam, 2007.

[CLR08] Jean-François Cordeau, Gilbert Laporte, and Stefan Ropke. Recent Models and Algorithms for One-to-One Pickup and Delivery Problems. In *The Vehicle Routing Problem: Latest Advances and New Challenges*, Operations Research/Computer Science Interfaces Series, pages 327–357. Springer US, 2008.

[CLSV07] Jean-François Cordeau, Gilbert Laporte, Martin W. P. Savelsbergh, and Daniele Vigo. Vehicle Routing. In Cynthia Barnhart and Gilbert Laporte, editors, *Handbooks in Operations Research and Management Science: Transportation*, volume 14, pages 367–428. Elsevier, Amsterdam, 2007.

[CLZ99] Giacomo Cabri, Letizia Leonardi, and Franco Zambonelli. Reactive Tuple Spaces for Mobile Agent Coordination. In *Proceedings of the 2nd International Workshop on Mobile Agents (MA 1998)*, volume 1477 of *Lecture Notes in Computer Science*, pages 237–248. Springer, 1999.

[CLZ00] Giacomo Cabri, Letizia Leonardi, and Franco Zambonelli. Mobile-Agent Coordination Models for Internet Applications. *Computer*, 33(2):82–89, 2000.

[CM95] James P. Crutchfield and Melanie Mitchell. The evolution of emergent computation. *Proceedings of the National Academy of Sciences*, 92(23):10742–10746, 1995.

[CMMS+07] Emre Cakar, Moez Mnif, Christian Müller-Schloer, Urban Richter, and Hartmut Schmeck. Towards a Quantitative Notion of Self-organisation. In *Proceedings of 2007 IEEE Congress on Evolutionary Computation (CEC 2007)*, pages 4222–4229. IEEE, 2007.

[CNF01] Gianpaolo Cugola, Elisabetta Di Nitto, and Alfonso Fuggetta. The JEDI Event-Based Infrastructure and Its Application to the Development of the OPSS WFMS. *IEEE Transactions on Software Engineering*, 27(9):827–850, 2001.

[Com02] Computing Research Association. Grand Research Challenges in Information Systems. http://www.cra.org/reports/gc.systems.pdf, 2002.

[COR08] CORMAS. http://cormas.cirad.fr/indexeng.htm, 2008.

[COZ99] Paolo Ciancarini, Andrea Omicini, and Franco Zambonelli. Coordination Technologies for Internet Agents. *Nordic Journal of Computing*, 6(3):215–240, 1999.

[COZ00] Paolo Ciancarini, Andrea Omicini, and Franco Zambonelli. Multiagent System Engineering: The Coordination Viewpoint. In *Proceedings of the 6th International Workshop on Intelligent Agents VI, Agent Theories, Architectures, and Languages (ATAL 1999)*, volume 1757 of *Lecture Notes in Computer Science*, pages 250–259. Springer, 2000.

[CP06] Gianpaolo Cugola and Gian Pietro Picco. REDS: A Reconfigurable Dispatching System. In *Proceedings of the 6th International Workshop on Software Engineering and Middleware (SEM 2006)*, pages 9–16. ACM, 2006.

[CR97] Paolo Ciancarini and Davide Rossi. Jada - Coordination and Communication for Java Agents. In *Proceedings of the 2nd International Workshop on Mobile Object Systems - Towards the Programmable Internet (MOS 1996), Selected Presentations and Invited Papers*, volume 1222 of *Lecture Notes in Computer Science*, pages 213–226. Springer-Verlag, 1997.

[CR03] Lauren Clement and Radhika. Self-Assembly and Self-Repairing Topologies. In *Proceedings of the Workshop on Adaptability in Multi-Agent Systems, First RoboCup Australian Open*, 2003.

[Cru94a] James P. Crutchfield. Is Anything Ever New? Considering Emergence. In G. Cowan, D. Pines, and D. Melzner, editors, *Complexity: Metaphors, Models, and Reality*, volume 19 of *Santa Fe Institute Series in the Sciences of Complexity*, pages 479–497. Addison-Wesley, Redwood City, 1994.

[Cru94b] James P. Crutchfield. The Calculi of Emergence: Computation, Dynamics, and Induction. *Physica D*, 75:11–54, 1994.

[CRW01] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, 2001.

[CS91] Scott Camazine and James Sneyd. A model of collective nectar source selection by honey bees: Self-organization through simple rules. *Journal of Theoretical Biology*, 149(4):547–571, 1991.

[CSC04] Massimo Cossentino, Luca Sabatucci, and Antonio Chella. Patterns Reuse in the PASSI Methodology. In Andrea Omicini, Paolo Petta, and Jeremy Pitt, editors, *Proceedings of the 4th International Workshop on Engineering Societies in the Agents World IV (ESAW 2003), Revised Selected and Invited Papers*, volume 3071 of *Lecture Notes in Computer Science*, pages 294–310, 2004.

[CSLG06] Chin Soon Chong, Appa Iyer Sivakumar, Malcolm Yoke Hean Low, and Kheng Leng Gay. A bee colony optimization algorithm to job shop scheduling. In *Proceedings of the 38th Winter Simulation Conference (WSC 2006)*, pages 1954–1961. Winter Simulation Conference, 2006.

[CTV+98] Paolo Ciancarini, Robert Tolksdorf, Fabio Vitali, Davide Rossi, and Andreas Knoche. Coordinating Multiagent Applications on the WWW: A Reference Architecture. *IEEE Transactions on Software Engineering*, 24(5):362–375, 1998.

[CUBT05] Hans Czap, Rainer Unland, Cherif Branki, and Huaglory Tianfield, editors. *Self-Organization and Autonomic Informatics (I), Proceedings of the SOAS 2005 Conference, Glasgow, UK*, volume 135 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2005.

[CW64] G. Clarke and J. W. Wright. Scheduling of Vehicles from a Central Depot to a Number of Delivery Points. *Operations Research*, 12(4):568–581, 1964.

[CWO91] Sarah A. Corbet, Ingrid H. Williams, and Juliet L. Osborne. Bees and the pollination of crops and wild flowers in the European Community. *Bee World*, 72:47–59, 1991.

[DAGP90] Jean-Louis Deneubourg, Serge Aron, Simon Goss, and Jacques M. Pasteels. The Self-organizing Exploratory Pattern of the Argentine Ant. *Journal of Insect Behavior*, 3(2):159–168, March 1990.

[Dak65] Robert J. Dakin. A tree-search algorithm for mixed integer programming problems. *The Computer Journal*, 8(3):220–255, 1965.

[DB07] Richard C. Dorf and Robert H. Bishop. *Modern Control Systems*. Prentice Hall, 11th edition, 2007.

[DC05] Klaus Dorer and Monique Calisti. An adaptive solution to dynamic transport optimization. In *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005)*, pages 45–51. ACM, 2005.

[DD98] Gianni Di Caro and Marco Dorigo. Two Ant Colony Algorithms for Best-effort Routing in Datagram Networks. In *Proceedings of the 10th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 1998)*, pages 541–546, 1998.

[DDE+02] Guy Desaulniers, Jacques Desrosiers, Andreas Erdmann, Marius M. Solomon, and François Soumis. VRP with Pickup and Delivery. In Paolo Toth and Daniele Vigo, editors, *The Vehicle Routing Problem*, SIAM Monographs on Discrete Mathematics and Applications, pages 225–242. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2002.

[DDF+06] Simon Dobson, Spyros Denazis, Antonio Fernández, Dominique Gaïti, Erol Gelenbe, Fabio Massacci, Paddy Nixon, Fabrice Saffre, Nikita Schmidt, and Franco Zambonelli. A Survey of Autonomic Communications. *ACM Transactions on Autonomous and Adaptive Systems*, 1(2):223–259, 2006.

[DDKB10] Florian Dötsch, Jörg Denzinger, Holger Kasinger, and Bernhard Bauer. Decentralized Real-time Control of Water Distribution Networks Using Self-organizing Multi-Agent Systems. In *Proceedings of the 4th IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2010)*, pages 223–232. IEEE, 2010.

[DDS91] Yvan Dumas, Jacques Desrosiers, and François Soumis. The pickup and delivery problem with time windows. *European Journal of Operational Research*, 54(1):7–22, September 1991.

[De 07] Tom De Wolf. *Analysing and Engineering Self-organising Emergent Applications*. PhD thesis, Department of Computer Science, K.U.Leuven, Leuven, Belgium, May 2007.

[Des85] René Descartes. Discours de la Méthode pour bien conduire sa raison, et chercher la vérité dans les sciences (1637). In *The Philosophical Writings of Descartes*, volume 1, pages 111–151. Cambridge University Press, 1985.

[DFB+01] Pat Dallard, Tony Fitzpatrick, Sophie Le Bourva, Angus Low, Roger Ridsdill Smith, and Michael Willford. The London Millennium Footbridge. *Structural Engineer*, 79(22):17–35, 2001.

[DFT90] John Doyle, Bruce Francis, and Allen Tannenbaum. *Feedback Control Theory*. Macmillan Publishing Co., 1990.

[DG89] Jean-Louis Deneubourg and Simon Goss. Collective patterns and decision-making. *Ethology, Ecology & Evolution*, 1:295–311, 1989.

[DGH+87] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the 6th annual ACM Symposium on Principles of distributed computing (PODC 1987)*, pages 1–12. ACM, 1987.

[DH02] Guy A. Dumont and Mihai Huzmezan. Concepts, Methods and Techniques in Adaptive Control. In *Proceedings of the 2002 American Control Conference (ACC 2002)*, pages 1137–1150. IEEE, 2002.

[DH04] Tom De Wolf and Tom Holvoet. Emergence Versus Self-Organisation: Different Concepts but Promising When Combined. In Sven Brueckner, Giovanna Di Marzo Serugendo, Anthony Karageorgos, and Radhika Nagpal, editors, *Engineering Self-Organising Systems*, volume 3464 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2004.

[DH05] Tom De Wolf and Tom Holvoet. Towards a Methodolgy for Engineering Self-Organising Emergent Systems. *Self-Organization and Autonomic Informatics (I), Frontiers in Artificial Intelligence and Applications*, 135:18–34, 2005.

[DH06] Tom De Wolf and Tom Holvoet. A Catalogue of Decentralised Coordination Mechanisms for Designing Self-Organising Emergent Applications. Report CW 458, Department of Computer Science, K.U.Leuven, Leuven, Belgium, 2006.

[DH07a] Tom De Wolf and Tom Holvoet. A Taxonomy for Self-* Properties in Decentralised Autonomic Computing. In Manish Parashar and Salim Hariri, editors, *Autonomic Computing: Concepts, Infrastructure, and Applications*, pages 101–120. CRC Press, 2007.

[DH07b] Tom De Wolf and Tom Holvoet. Design Patterns for Decentralised Coordination in Self-organising Emergent Systems. In Sven Brueckner, Salima Hassas, Márk Jelasity, and Daniel Yamins, editors, *Proceedings of the 4th International Workshop on Engineering Self-Organising Applications (ESOA 2006)*, volume 4335 of *Lecture Notes in Computer Science*, pages 28–49. Springer, 2007.

[DHP+05] Yixin Diao, Joseph L. Hellerstein, Sujay Parekh, Rean Griffith, Gail E. Kaiser, and Dan Phung. A Control Theory Foundation for Self-managing Computing Systems. *IEEE Journal on Selected Areas in Communications*, 23(12):2213–2222, 2005.

[DHP+06] Yixin Diao, Joseph L. Hellerstein, Sujay S. Parekh, Hidayatullah Shaikh, and Maheswaran Surendra. Controlling Quality of Service in Multi-Tier Web Applications. In *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems (ICDCS 2006)*, pages 25–32. IEEE Computer Society, 2006.

[DKL+08] Rajarshi Das, Jeffrey O. Kephart, Charles Lefurgy, Gerald Tesauro, David W. Levine, and Hoi Chan. Autonomic multi-agent management of power and performance in data centers. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, pages 107–114. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[DLD97] Peter Dömel, Anselm Lingnau, and Oswald Drobnik. Mobile Agent Interaction in Heterogeneous Environments. In *Proceedings of the 1st International Workshop on Mobile Agents (MA 1997)*, volume 1219 of *Lecture Notes in Computer Science*, pages 136–148. Springer, 1997.

[DN08] Dipankar Dasgupta and Fernando Nino. *Immunological Computation: Theory and Applications*. Auerbach Publications, 2008.

[dNFP98] Rocco de Nicola, Gian Luigi Ferrari, and Rosario Pugliese. KLAIM: A Kernel Language for Agents Interaction and Mobility. *IEEE Transactions on Software Engineering*, 24(5):315–330, 1998.

[DNO97] Enrico Denti, Antonio Natali, and Andrea Omicini. Programmable Coordination Media. In *Proceedings of the 2nd International Conference on Coordination Languages and Models (COORDINATION 1997)*, volume 1282 of *Lecture Notes In Computer Science*, pages 274–288. Springer, 1997.

[Dor87] Peter Dorato. A Historical Review of Robust Control. *IEEE Control Systems Magazine*, 7(2):44–47, 1987.

[Dot86] R. L. Doty. Odor-guided behavior in mammals. *Cellular and Molecular Life Sciences*, 42(3):257–271, 1986.

[Döt10] Florian Dötsch. Real-time Control of Water Distribution Networks Using Self-Organizing Emergent Multi-Agent Systems. Diploma thesis, University of Augsburg, May 2010.

[DPH07] Paul Davidsson, Jan A. Persson, and Johan Holmgren. On the Integration of Agent-Based and Mathematical Optimization Techniques. In *Proceedings of the 1st KES International Symposium on Agent and Multi-Agent Systems (KES-AMSTA 2007)*, pages 1–10, Berlin, Heidelberg, 2007. Springer-Verlag.

[DR59] George B. Dantzig and John H. Ramser. The Truck Dispatching Problem. *Management Science*, 6(1):80–91, 1959.

[DS88] Marcel Dicke and Maurice W. Sabelis. Infochemical Terminology: Based on Cost-Benefit Analysis Rather than Origin of Compounds? *Functional Ecology*, 2(2):131–139, 1988.

[DS04] Marco Dorigo and Thomas Stützle. *Ant Colony Optimization*. MIT Press, 2004.

[Duf76] S. S. Duffey. Arthropod allomones: chemical effronteries and antagonists. In *Proceedings of the 15th International Congress of Entomology*, pages 323–394, 1976.

[DvdHT02] Eric M. Dashofy, André van der Hoek, and Richard N. Taylor. Towards architecture-based self-healing systems. In *Proceedings of the 1st Workshop on Self-healing systems (WOSS 2002)*, pages 21–26, New York, NY, USA, 2002. ACM.

[DvPdB03] Marcel Dicke, Remco M.P. van Poeckea, and Jetske G. de Boera. Inducible Indirect Defence of Plants: From Mechanisms to Ecological Functions. *Basic and Applied Ecology*, 4(1):27–42, 2003.

[DWK01] Dwight Deugo, Michael Weiss, and Elizabeth Kendall. Reusable patterns for agent coordination. In Andrea Omicini, Franco Zambonelli, Matthias Klusch, and Robert Tolksdorf, editors, *Coordination of Internet agents: models, technologies, and applications*, pages 347–368. Springer-Verlag, 2001.

[DWS01] Scott A. DeLoach, Mark F. Wood, and Clint H. Sparkman. Multiagent Systems Engineering. *International Journal of Software Engineering and Knowledge Engineering*, 11(3):231–258, 2001.

[Dys98] George B. Dyson. *Darwin Among The Machines: The Evolution Of Global Intelligence*. Basic Books, 1998.

[DZKS06] M. Bernardine Dias, Robert Zlot, Nidhi Kalra, and Anthony Stentz. Market-Based Multirobot Coordination: A Survey and Analysis. *Proceedings of the IEEE*, 94(7):1257–1270, July 2006.

[EB04] Bruce Edmonds and Joanna J. Bryson. The Insufficiency of Formal Design Methods: The Necessity of an Experimental Approach - for the Understanding and Control of Complex MAS. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, pages 938–945. IEEE Computer Society, 2004.

[Edm05] Bruce Edmonds. Using the Experimental Method to Produce Reliable Self-Organised Systems. In *Engineering Self-Organising Systems, Methodologies and Applications (ESOA 2004)*, volume 3464 of *Lecture Notes in Computer Science*, pages 84–99. Springer, 2005.

[EFGK03] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys*, 35(2):114–131, 2003.

[Enc08] Encyclopædia Britannica. Autonomic Nervous System, 2008.

[ERA+03] Torsten Eymann, Michael Reinicke, Oscar Ardaiz, Pau Artigas, Felix Freitag, and Leandro Navarro. Self-Organizing Resource Allocation for Autonomic Networks. In *Proceedings of the 14th International Workshop on Database and Expert Systems Applications (DEXA 2003)*, pages 656–660. IEEE Computer Society, 2003.

[ERRAA04] Marc Esteva, Bruno Rosell, Juan A. Rodriguez-Aguilar, and Josep Ll. Arcos. AMELI: An Agent-Based Middleware for Electronic Institutions. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, pages 236–243. IEEE Computer Society, 2004.

[ESB06] Greg Eisenhauer, Karsten Schwan, and Fabián E. Bustamante. Publish-Subscribe for High-Performance Computing. *IEEE Internet Computing*, 10(1):40–47, 2006.

[ETJ04] Cora Beatriz Excelente-Toledo and Nicholas R. Jennings. The Dynamic Selection of Coordination Mechanisms. *Autonomous Agents and Multi-Agent Systems*, 9(1-2):55–85, 2004.

[EU 10] EU Integrated Project 'MyCar' . http://www.mycar-project.eu, January 2010.

[EUC09] EUCAR. The Automotive Industry – Focus on future R&D Challenges. http://www.eucar.be/publications/EUCAR%20FOCUS%202009_Web.pdf, 2009.

[Eva48] Walter R. Evans. Graphical Analysis of Control Systems. *Transactions of the American Institute of Electrical Engineers*, 67:547–551, 1948.

[Fai98] Joe Faith. Why gliders don't exist: anti-reductionism and emergence. In *Proceedings of the 6th International Conference on Artificial Life (ALIFE 1998)*, pages 389–392. MIT Press, 1998.

[Far09] Muddassar Farooq. *Bee-Inspired Protocol Engineering.* Natural Computing Series. Springer, 2009.

[FAW+04] Charles B. Fenster, W. Scott Armbruster, Paul Wilson, Michele R. Dudash, and James D. Thomson. Pollination syndromes and floral specialization. *Annual Review of Ecology, Evolution, and Systematics*, 35:375–403, December 2004.

[FBDM96] Munehiro Fukuda, Lubomir F. Bic, Michael B. Dillencourt, and Fehmina Merchant. Intra- and Inter-Object Coordination with MESSENGERS. In *Proceedings of the 1st International Conference on Coordination Languages and Models (COORDINATION 1996)*, volume 1061 of *Lecture Notes In Computer Science*, pages 179–196. Springer, 1996.

[FBG96] Gert Florijn, Timo Besamusca1, and Danny Greefhorst. ARIADNE and HOPLa: Flexible Coordination of Collaborative Processes. In *Proceedings of the 1st International Conference on Coordination Languages and Models (COORDINATION 1996)*, volume 1061 of *Lecture Notes in Computer Science*, pages 197–214. Springer, 1996.

[FCG06] Bernard Feltz, Marc Crommelinck, and Philippe Goujon. *Self-organization and Emergence in Life Sciences*. Springer Netherlands, 2006.

[FECA04] Robert E. Filman, Tzilla Elrad, Siobhán Clarke, and Mehmet Aksit. *Aspect-Oriented Software Development*. Addison-Wesley Professional, 2004.

[FF05] Mark E. Femal and Vincent W. Freeh. Boosting Data Center Performance Through Non-Uniform Power Allocation. In *Proceedings of the 2nd International Conference on Automatic Computing (ICAC 2005)*, pages 250–261. IEEE Computer Society, 2005.

[FG97] Stan Franklin and Art Graesser. Is it an Agent, or Just a Program?: A Taxonomy for Autonomous Agents. In *Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages (ECAI 1996)*, volume 1193 of *Lecture Notes In Computer Science*, pages 21–35. Springer, 1997.

[FHA99] Eric Freeman, Susanne Hupfer, and Ken Arnold. *JavaSpaces Principles, Patterns, and Practice*. Addison-Wesley Longman Ltd., Essex, UK, 1999.

[FHK+07] Truls Flatberg, Geir Hasle, Oddvar Kloster, Eivind J. Nilssen, and Atle Riise. Dynamic and Stochastic Vehicle Routing in Practice. In *Dynamic Fleet Management*, volume 38 of *Operations Research/-Computer Science Interfaces Series*, pages 41–63. Springer US, 2007.

[FINZ05] Alessandro Farinelli, Luca Iocchi, Daniele Nardi, and Vittorio A. Ziparo. Task Assignment with Dynamic Perception and Constrained Tasks in a Multi-Robot System. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation (ICRA 2005)*, pages 1523–1528, 2005.

[FIP02a] FIPA. FIPA Abstract Architecture Specification. http://www.fipa.org/specs/fipa00001/, 2002.

[FIP02b] FIPA. FIPA Communicative Act Library Specification. http://www.fipa.org/specs/fipa00037/, 2002.

[FIP02c] FIPA. FIPA Contract Net Interaction Protocol Specification. http://www.fipa.org/specs/fipa00029/SC00029H.pdf, 2002.

[Fis95] Marshall Fisher. Vehicle Routing. In M.O. Ball, T.L. Magnanti, C.L. Monma, and G.L. Nemhauser, editors, *Handbooks in Operations Research and Management Science: Network Routing*, volume 8, pages 1–33. North-Holland, Amsterdam, 1995.

[Fis96] Klaus Fischer. Cooperative Transportation Scheduling: An Application Domain for DAI. *Applied Artificial Intelligence*, 10(1):1–34, 1996.

[FKM94] Klaus Fischer, Norbert Kuhn, and Jörg P. Muller. Distributed, Knowledge-based, Reactive Scheduling of Transportation Tasks. In *Proceedings of the 10th Conference on Artificial Intelligence for Applications (AAAI 1994)*, pages 47–53. IEEE, 1994.

[Fla00] Gary William Flake. *The Computational Beauty of Nature: Computer Explorations of Fractals, Chaos, Complex Systems, and Adaptation*. MIT Press, 2000.

[FLPW86] J. Doyne Farmer, Alan Lapedes, Norman Packard, and Burton Wendro, editors. *Evolution, Games, and Learning: Models for Adaptation in Machines and Nature: Proceedings of the 5th Annual International Conference of the Center for Nonlinear Studies*, Los Alamos, New Mexico, USA, 1986. North-Holland.

[FMM94] Tim Finin, Richard Fritzson Don McKay, and Robin McEntire. KQML as an agent communication language. In *Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM 1994)*, pages 456–463. ACM, 1994.

[FMPS95] Klaus Fischer, Jörg P. Müller, Markus Pischel, and Darius Schier. A Model for Cooperative Transportation Scheduling. In *Proceedings of the 1st International Conference on Multiagent Systems (ICMAS 1995)*, pages 109–116. MIT Press, 1995.

[For90] Stephanie Forrest, editor. *Emergent Computation: Self-Organizing, Collective, and Cooperative Phenomena in Natural and Artificial Computing Networks: Proceedings of the 9th Annual International Conference of the Center for Nonlinear Studies*, Los Alamos, New Mexico, USA, 1990. North-Holland.

[FP79] Knut Faegri and Leendert Van Der Pijl. *The Principles of Pollination Ecology*. Pergamon Press, Oxford, 3. edition, September 1979.

[FPEN09] Gene F. Franklin, J. David Powell, and Abbas Emami-Naeini. *Feedback Control of Dynamic Systems*. Prentice Hall, 6th edition, 2009.

[FR06] Anke Fabri and Peter Recht. On dynamic pickup and delivery vehicle routing with several time windows and waiting times. *Transportation Research Part B: Methodological*, 40(4):335–350, 2006.

[Fre87] John B. Free. *Pheromones of Social Bees*. Comstock Publishing, Ithaca, N.Y., 1987.

[Fro05] Jochen Fromm. Types and Forms of Emergence. Technical report, University of Kassel, http://arxiv.org/ftp/nlin/papers/0506/0506028.pdf, 2005.

[FU00] Nikolai M. Filatov and Heinz Unbehauen. Survey of Adaptive Dual Control Methods. *IEEE Proc. Control Theory Applications*, 147(1):118–128, 2000.

[GADP89] Simon Goss, Serge Aron, Jean-Louis Deneubourg, and Jacques M. Pasteels. Self-organized Shortcuts in the Argentine Ant. *Naturwissenschaften*, 76:579–581, 1989.

[Gat98] Erann Gat. Three-layer Architectures. In David Kortenkamp, R. Peter Bonasso, and Robin Murphy, editors, *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems*, pages 195–210. AAAI Press, 1998.

[GC92] David Gelernter and Nicholas Carriero. Coordination languages and their significance. *Communications of the ACM*, 35(2):97–107, 1992.

[GCGC08] Marie-Pierre Gleizes, Valérie Camps, Jean-Pierre Georgé, and Davy Capera. Engineering Systems Which Generate Emergent Functionalities. In *Proceedings of the International Workshop on Engineering Environment-Mediated Multi-Agent Systems (EEMMAS 2007), Selected Revised and Invited Papers*, Lecture Notes In Artificial Intelligence, pages 58–75. Springer-Verlag, 2008.

[GCH+04] David Garlan, Shang-Wen Cheng, An-Cheng Huang, Bradley Schmerl, and Peter Steenkiste. Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure. *Computer*, 37(10):46–54, 2004.

[GCS03] David Garlan, Shang-Wen Cheng, and Bradley Schmerl. Increasing System Dependability through Architecture-Based Self-Repair. In *Architecting Dependable Systems*, volume 2677 of *Lecture Notes in Computer Science*, pages 61–89. Springer, 2003.

[Gel85] David Gelernter. Generative Communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, 1985.

[Gel89] David Gelernter. Multiple Tuple Spaces in Linda. In *Proceedings of the Parallel Architectures and Languages Europe, Volume II: Parallel Languages (PARLE 1989)*, pages 20–27. Springer, 1989.

[GEL+06] Eli Gjørven, Frank Eliassen, Ketil Lund, Viktor S. Wold Eide, and Richard Staehli. Self-Adaptive Systems: A Middleware Managed Approach. In *Proceedings of the 2nd IEEE International Workshop on Self-Managed Networks, Systems, and Services (SelfMan 2006)*, volume 3996 of *Lecture Notes in Computer Science*, pages 15–27. Springer, 2006.

[Ger05] German Science Foundation. Priority Program 1183 Organic Computing. http://www.organic-computing.de/spp, 2005.

[Ger07] Carlos Gershenson. *Design and Control of Self-organizing Systems*. Phd thesis, Vrije Universiteit Brussel, 2007.

[GGH08] Cristina Gacek, Holger Giese, and Ethan Hadar. Friends or foes?: a conceptual analysis of self-adaptation and it change management. In *Proceedings of the 3rd International Workshop on Software Engineering for Adaptive and Self-managing Systems (SEAMS 2008)*, pages 121–128. ACM, 2008.

[GGLM03] Gianpaolo Ghiani, Francesca Guerriero, Gilbert Laporte, and Roberto Musmanno. Real-time vehicle routing: Solution concepts, algorithms and parallel computing strategies. *European Journal of Operational Research*, 151(1):1–11, 2003.

[GGPS06] Michel Gendreau, François Guertina, Jean-Yves Potvina, and René Séguin. Neighborhood search heuristics for a dynamic vehicle dispatching problem with pick-ups and deliveries. *Transportation Research Part C: Emerging Technologies*, 14(3):157–174, 2006.

[GGT07] Simon Garnier, Jacques Gautrais, and Guy Theraulaz. The Biological Principles of Swarm Intelligence. *Swarm Intelligence*, 1(1):3–31, 2007.

[GH03] Carlos Gershenson and Francis Heylighen. When Can we Call a System Self-organizing? In W. Banzhaf, T. Christaller, P. Dittrich, J. T. Kim, and J. Ziegler, editors, *Proceedings of the 7th European Conference on Advances in Artificial Life (ECAL 2003)*, volume 2801 of *Lecture Notes of Artificial Intelligence*, pages 606–614. Springer, 2003.

[GHJV94] Erich Gamma, Richard Helm, Ralph Johnsona, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, October 1994.

[Gil02] Frank E. Gillett. Organic IT, April 2002.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[GJVG99] M. A. Gibney, Nicholas R. Jennings, N. J. Vriend, and José-Marie Griffiths. Market-Based Call Routing in Telecommunications Networks Using Adaptive Pricing and Real Bidding. In S. Albayrak, editor, *Proceedings of the 3rd International Workshop on Intelligent Agents for Telecommunication Applications (IATA 1999)*, volume 1699, pages 46–61. Springer-Verlag, 1999.

[GKM02] Al Gillen, Dan Kusnetzky, and Scott McLarnon. The Role of Linux in Reducing the Cost of Enterprise Computing - IDC White Paper. Red Hat Incorporation, January 2002.

[GKMP04] Paolo Giorgini, Manuel Kolp, John Mylopoulos, and Marco Pistore. The Tropos Methodology: An Overview. In Federico Bergenti, Marie-Pierre Gleizes, and Franco Zambonelli, editors, *Methodologies And Software Engineering For Agent Systems*. Kluwer Academic Publishing, 2004.

[GL05] Robert J. Gegear and Terence M. Laverty. Flower constancy in bumblebees: a test of the trait variability hypothesis. *Animal Behaviour*, 69:939–949, 2005.

[GL08] Irina Gribkovskaia and Gilbert Laporte. One-to-Many-to-One Single Vehicle Pickup and Delivery Problems. In *The Vehicle Routing Problem: Latest Advances and New Challenges*, Operations Research/Computer Science Interfaces Series, pages 359–377. Springer US, 2008.

[Gla98] Graham Glass. ObjectSpace Voyager - The Agent ORB for Java. In *Proceedings of the 2nd International Conference on Worldwide Computing and Its Applications (WWCA 1998)*, volume 1368 of *Lecture Notes In Computer Science*, pages 38–55. Springer, 1998.

[GLS96] Michel Gendreau, Gilbert Laporte, and René Séguin. Stochastic vehicle routing. *European Journal of Operational Research*, 88(1):3–12, January 1996.

[GM74] Billy E. Gillett and Leland R. Miller. A Heuristic Algorithm for the Vehicle-Dispatch Problem. *Operations Research*, 22(2):340–349, 1974.

[GMQT09] Gianpaolo Ghiani, Emanuele Mannia, Antonella Quarantaa, and Chefi Triki. Anticipatory algorithms for same-day courier dispatching. *Transportation Research Part E: Logistics and Transportation Review*, 45(1):96–106, 2009.

[GNO+08] Matthias Güdemann, Florian Nafz, Frank Ortmeier, Hella Seebach, and Wolfgang Reif. A Specification and Construction Paradigm for Organic Computing Systems. In Sven Brueckner, Paul Robertson, and Umesh Bellur, editors, *Proccedings of the 2nd IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2008)*. IEEE Computer Society, 2008.

[GNV04] Kai Gutenschwager, Christian Niklaus, and Stefan Voß. Dispatching of an Electric Monorail System: Applying Metaheuristics to an Online Pickup and Delivery Problem. *Transportation Science*, 38(4):434–446, 2004.

[Gol99] Jeffrey Goldstein. Emergence as a Construct: History and Issues. *Emergence*, 1(1):49–72, 1999.

[GP71] Paul Glansdorff and Ilya Prigogine. *Thermodynamic Theory of Structure, Stability and Fluctuations*. Wiley, 1971.

[GP98] Michel Gendreau and Jean-Yves Potvin. Dynamic Vehicle Routing and Dispatching. In Teodor Gabriel Crainic and Gilbert Laporte, editors, *Fleet Management and Logistics*, pages 115–126. Kluwer, Boston, 1998.

[Gra59] Pierre-Paul Grassé. La reconstruction du nid et les coordinations individuelles chez bellicositermes natalensis et cubitermes sp. la theorie de la stigmergie: Essai d'interpretation du comportement des Termites Constructeurs. *Insectes Sociaux*, 6:41–80, 1959.

[GRW08] Bruce Golden, S. Raghavan, and Edward Wasil, editors. *The Vehicle Routing Problem: Latest Advances and New Challenges*, volume 43 of *Operations Research/Computer Science Interfaces Series*. Springer US, 2008.

[GS02] David Garlan and Bradley Schmerl. Model-based adaptation for self-healing systems. In *Proceedings of the 1st Workshop on Self-healing Systems (WOSS 2002)*, pages 27–32. ACM, 2002.

[GSM+95] Kenneth J. Goldman, Bala Swaminathan, T. Paul McCartney, Michael D. Anderson, and Ram Sethuraman. The Programmers' Playground: I/O Abstraction for User-Configurable Distributed Applications. *IEEE Transactions on Software Engineering*, 21(9):735–746, 1995.

[Gut91] Howard Gutowitz. *Cellular Automata: Theory and Experiment (Special Issues of Physica D)*. MIT Press, 1991.

[GVCO08] Luca Gardelli, Mirko Viroli, Matteo Casadei, and Andrea Omicini. Designing self-organising environments with agents and artefacts: a simulation-driven approach. *International Journal of Agent-Oriented Software Engineering*, 2(2):171–195, 2008.

[GVO07] Luca Gardelli, Mirko Viroli, and Andrea Omicini. Design Patterns for Self-organizing Multiagent Systems. In Hans-Dieter Burkhard, Gabriela Lindemann, Rineke Verbrugge, and László Zsolt Varga, editors, *Proceedings of the 5th International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS 2007)*, volume 4696 of *Lecture Notes in Computer Science*, pages 123–132. Springer, 2007.

[HA07] MyungJoo Ham and Gul Agha. Market-based Coordination Strategies for Large-scale Multi-Agent Systems. *Communications of SIWN*, 2(1):126–131, 2007.

[Hak77] Hermann Haken. *Synergetics: An introduction: Nonequilibrium Phase Transitions and Self-Organization in Physics, Chemistry, and Biology*. Springer-Verlag, Berlin, 1977.

[Hak81] Hermann Haken. *The Science of Structure: Synergetics*. Van Nostrand Reinhold, 1981.

[HAK94] Béat Hirsbrunner, Marc Aguilar, and Oliver Krone. CoLa: a coordination language for massive parallelism. In *Proceedings of the 13th Annual ACM Symposium on Principles of Distributed Computing (PODC 1994)*, page 384. ACM, 1994.

[Hak06] Hermann Haken. *Information and Self-organization: A Macroscopic Approach to Complex Systems*. Springer Verlag, Berlin, 2006.

[Hal06] David Hales. Choose Your Tribe! - Evolution at the Next Level in a Peer-to-Peer Network. In Sven Brueckner, Giovanna Di Marzo Serugendo, David Hales, and Franco Zambonelli, editors, *Proceedings of the 3rd International Workshop on Engineering Self-Organising Systems (ESOA 2005), Revised Selected Papers*, volume 3910 of *Lecture Notes in Computer Science*, pages 61–74. Springer, 2006.

[Har75] John A. Hartigan. *Clustering Algorithms*. John Wiley & Sons, 1975.

[Hay98] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 2nd edition, 1998.

[HD09] Sascha Häckel and Patrick Dippold. The bee colony-inspired algorithm (bcia): a two-stage approach for solving the vehicle routing problem with time windows. In *Proceedings of the 11th Annual Conference on Genetic and evolutionary computation (GECCO 2009)*, pages 25–32. ACM, 2009.

[HDPT04] Joseph L. Hellerstein, Yixin Diao, Sujay Parekh, and Dawn M. Tilbury. *Feedback Control of Computing Systems*. Wiley-IEEE Press, 2004.

[Hew07] Hewlett-Packard Development Company. HP Adaptive Infrastructure Whitepaper. http://h20195.www2.hp.com/PDF/4AA1-4757ENW.pdf, September 2007.

[Hew08] Hewlett-Packard Development Company. HP Adaptive Infrastructure. http://www.hp.com/go/ai, May 2008.

[Hey03] Francis Heylighen. The Science of Self-Organization and Adaptivity. In L. Douglas Kiel, editor, *Knowledge Management, Organizational Intelligence and Learning, and Complexity*, The Encyclopedia of Life Support Systems. EOLSS, 2003.

[HG03] Francis Heylighen and Carlos Gershenson. The Meaning of Self-Organization in Computing. *IEEE Intelligent Systems*, 18(4):72–75, 2003.

[HGB10] Regina Hebig, Holger Giese, and Basil Becker. Making Control Loops Explicit When Architecting Self-Adaptive Systems. In *Proceedings of the 2nd International Workshop on Self-Organizing Architectures (SOAR 2010)*, pages 22–28. ACM, 2010.

[Hir06] Katsutoshi Hirayama. Distributed Lagrangean relaxation protocol for the generalized mutual assignment problem. In *Proceedings of*

the 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006), pages 890–892. ACM, 2006.

[HJ01] Francis Heylighen and Cliff Joslyn. Cybernetics and Second-Order Cybernetics. *Encyclopedia of Physical Science & Technology*, 4:155–170, 2001.

[HM05] Markus C. Huebscher and Julie A. McCann. Using real-time dependability in adaptive service selection. In *Proceedings of the Joint International Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services (ICAS-ICNS 2005)*, pages 76–81. IEEE Computer Society, 2005.

[HM08] Markus C. Huebscher and Julie A. McCann. A survey of autonomic computing—degrees, models, and applications. *ACM Computing Surveys*, 40(3):1–28, 2008.

[HMG05] Klaus Herrmann, Gero Mühl, and Kurt Geihs. Self-Management: The Solution to Complexity or Just Another Problem? *IEEE Distributed Systems Online*, 6(1):1–1, 2005.

[HMH07] Markus C. Huebscher, Julie A. McCann, and Asher Hoskins. Context as autonomic intelligence in a ubiquitous computing environment. *International Journal of Internet Protocol Technology*, 2(1):30–39, 2007.

[HNR68] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[Hol92] John H. Holland. *Adaptation in natural and artificial systems.* MIT Press, Cambridge, MA, USA, 1992.

[Hol93] John H. Holland. The Effect of Labels (Tags) on Social Interactions. Technical report SFI Working Paper 93-10-064, Santa Fe Institute, Santa Fe, NM, 1993.

[Hol96] Alexandra Holzbacher. A Software Environment for Concurrent Coordinated Programming. In *Proceedings of the 1st International Conference on Coordination Languages and Models (COORDINATION 1996)*, volume 1061 of *Lecture Notes in Computer Science*, pages 249–266. Springer, 1996.

[Hol98] John H. Holland. *Emergence: From Chaos to Order.* Addison-Wesley, Redwood City, California, 1998.

[Hol02] John H. Holland. Echo. http://www.santafe.edu/~pth/echo, 2002.

[Hor01] Paul Horn. Autonomic Computing: IBM's Perspective on the State of Information Technology, October 2001.

[HP04] Pieter Jan't Hoen and Johannes A. La Poutré. A Decommitment Strategy in a Competitive Multi-agent Transportation Setting. In *Proceedings of the Agent-Mediated Electronic Commerce V Workshop (AAMAS 2003), Revised Selected Papers*, volume 3048 of *Lecture Notes in Computer Science*, pages 56–72. Springer, 2004.

[HPD09] Johan Holmgren, Jan A. Persson, and Paul Davidsson. Agent-Based Dantzig-Wolfe Decomposition. In *Proceedings of the 3rd KES International Symposium on Agent and Multi-Agent Systems: Technologies and Applications (KES-AMSTA 2009)*, volume 5559 of *Lecture Notes in Computer Science*, pages 754–763. Springer, 2009.

[HR85] Barbara Hayes-Roth. A blackboard architecture for control. *Artificial Intelligence*, 26(3):251–321, 1985.

[HRS91] Maritta Heisel, Wolfgang Reif, and Werner Stephan. Formal Software Development in the KIV System. In M. R. Lowry and R. D. McCartney, editors, *Automating Software Design*, pages 547–574. AAAI press, Menlo Park, CA, 1991.

[HS99a] Geoffrey Hinton and Terrence J. Sejnowski, editors. *Unsupervised Learning: Foundations of Neural Computation*. Computational Neuroscience. MIT Press, Cambridge, Massachusetts, 1999.

[HS99b] Michael N. Huhns and Larry M. Stephens. Multiagent Systems and Societies of Agents. In Gerhard Weiss, editor, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, pages 79–120. MIT Press, 1999.

[HS06] Michael G. Hinchey and Roy Sterritt. Self-Managing Software. *IEEE Computer*, 39(2):107–109, 2006.

[HSM+07] Tian He, John A. Stankovica, Michael Marleya, Chenyang Lu, Ying Lu, Tarek Abdelzaher, Sang Sona, and Gang Tao. Feedback control-based dynamic resource management in distributed real-time systems. *Journal of Systems and Software*, 80(7):997–1004, 2007.

[HVKB04] Karuna Hadeli, Paul Valckenaers, Martin Kollingbaum, and Hendrik Van Brussel. Multi-agent coordination and control using stigmergy. *Computers in Industry*, 53(1):75–96, 2004.

[HW90] Bert Hölldobler and Edward O. Wilson. *The Ants*. Springer, 1990.

[Ház34] Harold L. Házen. Theory of Servomechanisms. *Journal of the Franklin Institute*, 218:543–580, 1934.

[IBM02] IBM. Autonomic Computing Initiative. http://www.research.ibm.com/autonomic/, 2002.

[IBM03] IBM. Autonomic Computing Whitepaper: An Architectural Blueprint for Autonomic Computing, April 2003.

[IBM06] IBM. Autonomic Computing Whitepaper: An Architectural Blueprint for Autonomic Computing, June 2006.

[IBM08] IBM Institute for Business Value. The GMA 2008 Logistics Survey. http://www.gmabrands.com/publications/GMALogisticsStudy2008.pdf, 2008.

[IBM09] IBM. WebSphere MQ. www.ibm.com/webspheremq, 2009.

[IKV98] IKV++ GmbH. GrassHopper, an intelligent mobile agent platform written in 100% pure Java, 1998.

[Int99] International Commission on Zoological Nomenclature. *International Code of Zoological Nomenclature*. International Trust for Zoological Nomenclature, fouth edition edition, 1999.

[JAD08a] JADE. Java Agent DEvelopment Framework. http://jade.tilab.com, 2008.

[Jad08b] Jadex. http://vsis-www.informatik.uni-hamburg.de/projects/jadex, 2008.

[JB05] Márk Jelasity and Özalp Babaoglu. T-Man: Gossip-Based Overlay Topology Management. In Sven Brueckner, Giovanna Di Marzo Serugendo, David Hales, and Franco Zambonelli, editors, *Proceedings of the 3rd International Workshop on Engineering Self-Organising Systems (ESOA 2005), Revised Selected Papers*, volume 3910 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2005.

[JBL06] Márk Jelasity, Ozalp Babaoglu, and Robert Laddaga. Self-Management through Self-Organization. *IEEE Intelligent Systems*, 21(2):8–9, March/April 2006.

[JBR99] Ivar Jacobson, Grady Booch, and James Rumbaugh. *The unified software development process*. Addison Wesley, 1999.

[Jen93] Nicolas R. Jennings. Commitments and Conventions: The Foundation of Coordination in Multi-Agent Systems. *The Knowledge Engineering Review*, 8(3):223–250, 1993.

[Jen96] Nicolas R. Jennings. Coordination techniques for distributed artificial intelligence. In Gregory M. P. O'Hare and Nicolas R. Jennings, editors, *Foundations of distributed artificial intelligence*, pages 187–210. John Wiley & Sons, Inc., New York, NY, USA, 1996.

[Jen01] Nicholas R. Jennings. An Agent-Based Approach for Building Complex Software Systems. *Communications of the ACM*, 44(4):35–41, 2001.

[JF02] Brad Johanson and Armando Fox. The Event Heap: A Coordination Infrastructure for Interactive Workspaces. In *Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2002)*, page 83. IEEE Computer Society, 2002.

[JM93] Brian D. Jackson and E. David Morgan. Insect Chemical Communication: Pheromones and Exocrine Glands of Ants. *Chemoecology*, 4(3-4):125–144, Sep 1993.

[JMF99] Anil K. Jain, M. Narasimha Murty, and Patrick J. Flynn. Data Clustering: A Review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[JPS02] Thomas Juan, Adrian R. Pearce, and Leon Sterling. ROADMAP: extending the gaia methodology for complex open systems. In *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2002)*, pages 3–10. ACM, 2002.

[JSW98] Nicholas R. Jennings, Katia Sycara, and Michael Wooldridge. A Roadmap of Agent Research and Development. *Autonomous Agents and Multi-Agent Systems*, 1(1):7–38, 1998.

[KAH04] Nagarajan Kandasamy, Sherif Abdelwahed, and John P. Hayes. Self-Optimization in Computer Systems via On-Line Control: Application to Power Management. In *Proceedings of the 1st International Conference on Autonomic Computing (ICAC 2004)*, pages 54–61. IEEE Computer Society, 2004.

[Kau96] Stuart Kauffman. *At Home in the Universe: the Search for the Laws of Self-Organization and Complexity*. Oxford University Press, 1996.

[KB60a] Rudolf E. Kálmán and John E. Bertram. Control System Analysis and Design via the Second Method of Lyapunov: I. Continuous-time Systems. *Transactions of the ASME: Journal of Basic Engineering*, 82:371–393, 1960.

[KB60b] Rudolf E. Kálmán and John E. Bertram. Control System Analysis and Design via the Second Method of Lyapunov: II. Discrete Time Systems. *Transactions of the ASME: Journal of Basic Engineering*, 82:394–400, 1960.

[KB96] Paul Klint and Jan A. Bergstra. The TOOLBUS Coordination Architecture. In *Proceedings of the 1st International Conference on Coordination Languages and Models (COORDINATION 1996)*, volume 1061 of *Lecture Notes in Computer Science*, pages 75–88. Springer, 1996.

[KB06a] Holger Kasinger and Bernhard Bauer. Beyond Swarm Intelligence: Building Self-Managing Systems Based on Pollination. In Christian Hochberger and Rüdiger Liskowsky, editors, *Informatik 2006 - Informatik für Menschen, Band 1, Beiträge der 36. Jahrestagung der Gesellschaft für Informatik e.V. (GI)*, volume 93 of *Lecture Notes in Informatics*, pages 169–176. GI, 2006.

[KB06b] Holger Kasinger and Bernhard Bauer. Pollination - A Biologically Inspired Paradigm for Self-Managing Systems. *Journal of International Transactions on Systems Science and Applications*, 2(2):147–156, September 2006.

[KB06c] Holger Kasinger and Bernhard Bauer. The Utility of Pollination for Autonomic Computing. In Yi Pan, Franz Rammig, Hartmut Schmeck, and Mauricio Solar, editors, *Biologically Inspired Cooperative Computing*, volume 216 of *IFIP International Federation for Information Processing*, pages 55–64. Springer Boston, 2006.

[KB07] Dervis Karaboga and Bahriye Basturk. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. *Jornal of Global Optimization*, 39(3):459–471, 2007.

[KBD08] Holger Kasinger, Bernhard Bauer, and Jörg Denzinger. The Meaning of Semiochemicals to the Design of Self-Organizing Systems. In Sven Brueckner, Paul Robertson, and Umesh Bellur, editors, *Proceedings of the 2nd IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2008)*, pages 139–148. IEEE Computer Society, 2008.

[KBE99] Mieczyslaw M. Kokar, Kenneth Baclawski, and Yonet A. Eracar. Control theory-based foundations of self-controlling software. *IEEE Intelligent Systems*, 14:37–45, 1999.

[KC03] Jeffrey O. Kephart and David M. Chess. The Vision of Autonomic Computing. *IEEE Computer*, 36(1):41–50, January 2003.

[KD06] Jordan Kidney and Jörg Denzinger. Testing the Limits of Emergent Behavior in MAS Using Learning of Cooperative Behavior. In *Proceedings of the 17th European Conference on Artificial Intelligence (ECAI 2006)*, volume 141 of *Frontiers in Artificial Intelligence and Applications*, pages 260–264. IOS Press, 2006.

[KDB09] Holger Kasinger, Jörg Denzinger, and Bernhard Bauer. Decentralized Coordination of Homogeneous and Heterogeneous Agents by Digital Infochemicals. In *Proceedings of the 24th Annual ACM Symposium on Applied Computing (SAC 2009)*, pages 1223–1224. ACM Press, 2009.

[KE99] Robert Kohout and Kutluhan Erol. In-time agent-based vehicle routing with a stochastic improvement heuristic. In *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI 1999)*, pages 864–869. AAAI Press, 1999.

[Kep05] Jeffrey O. Kephart. Research Challenges of Autonomic Computing. In *Proceedings of the 27th International Conference on Software Engineering (ICSE 2005)*, pages 15–22. ACM Press, 2005.

[KG03] David Keil and Dina Q. Goldin. Modeling Indirect Interaction in Open Computational Systems. In *Proceedings of the 12th International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2003)*, pages 371–376. IEEE Computer Society, 2003.

[KG06] David Keil and Dina Goldin. Indirect Interaction in Environments for Multi-Agent Systems. In *Proceedings of the 2nd International Workshop on Environments for Multi-Agent Systems (E4MAS 2005), Selected Revised and Invited Papers*, volume 3830 of *Lecture Notes in Computer Science*, pages 68–87. Springer, 2006.

[KH07] Bithika Khargharia and Salim Hariri. Autonomic Power and Performance Management of Internet Data. In Manish Parashar and Salim Hariri, editors, *Autonomic Computing: Concepts, Infrastructure, and Applications*, pages 435–469. CRC Press, 2007.

[KH08] John Kavanagh and Wendy Hall. Grand Challenges in Computing Research. http://www.ukcrc.org.uk/grand_challenges/news/gccr08final.pdf, 2008.

[KHY08] Bithika Khargharia, Salim Hariri, and Mazin S. Yousif. Autonomic power and performance management for computing systems. *Cluster Computing*, 11(2):167–181, 2008.

[Kie96] Thilo Kielmann. Designing a Coordination Model for Open Systems. In *Proceedings of the 1st International Conference on Coordination Languages and Models (COORDINATION 1996)*, volume 1061 of *Lecture Notes in Computer Science*, pages 267–284. Springer, 1996.

[Kir04] Donald E. Kirk. *Optimal Control Theory: An Introduction.* Dover Publications, 2004.

[KKH+09] Dara Kusic, Jeffrey O. Kephart, James E. Hanson, Nagarajan Kandasamy, and Guofei Jiang. Power and performance management of virtualized computing environments via lookahead control. *Cluster Computing*, 12(1):1–15, 2009.

[KKPS98] Elizabeth A. Kendall, P. V. Murali Krishna, Chirag V. Pathak, and C. B. Suresh. Patterns of intelligent and mobile agents. In *Proceedings of the 2nd International Conference on Autonomous Agents (AGENTS 1998)*, pages 92–99. ACM, 1998.

[KL59] Peter Karlson and Martin Lüscher. "Pheromones" a new term for a class of biologically active substances. *Nature*, 183:155–156, 1959.

[KLS+03] Gabor Karsai, Akos Ledeczi, Janos Sztipanovits, Gabor Peceli, Gyula Simon, and Tamas Kovacshazy. An Approach to Self-adaptive Software Based on Supervisory Control. In *Proceedings of the 2nd International Workshop on Self-Adaptive Software: Applications (IWSAS 2001)*, volume 2614 of *Lecture Notes in Computer Science*, pages 77–92. Springer, 2003.

[KM07] Jeff Kramer and Jeff Magee. Self-Managed Systems: an Architectural Challenge. In *Proceedings of the 29th International Conference on Software Engineering (ICSE 2007), Future of Software Engineering (FOSE 2007)*, pages 259–268. IEEE Computer Society, 2007.

[KMF06] Alexander Keller and Jean-Philippe Martin-Flatin, editors. *Proceedings of the 2nd IEEE International Workshop on Self-Managed Networks, Systems, and Services (SelfMan 2006)*, volume 3996 of *Lecture Notes in Computer Science*. Springer, 2006.

[Koh84] Tuevo Kohonen. *Self-Organizing Maps and Associative Memory.* Springer-Verlag, Berlin, 1984.

[Koh01] Teuvo Kohonen. *Self-Organizing Maps.* Springer, Berlin, 3rd edition, 2001.

[KR02] Michael Köhler and Heiko Rölke. Modelling the Micro-Macro-Link: Towards a Sociologically Grounded Design of Multi-Agent Systems. In C. Jonker, G. Lindemann, and P. Panzarasa, editors, *Proceedings*

of the Workshop Modelling Artificial Societies and Hybrid Organization (MASHO 2002) at the 25th German Conference on Artificial Intelligence (KI 2002), pages 47–56, 2002.

[KRLM06] Majid Kazemian, Yoosef Ramezani, Caro Lucas, and Behzad Moshiri. Swarm Clustering Based on Flowers Pollination by Artificial Bees. In *Swarm Intelligence in Data Mining*, volume 34 of *Studies in Computational Intelligence*, pages 191–202. Springer, 2006.

[Kru96] Paul R. Krugman. *The Self-Organizing Economy*. Blackwell, Oxford, 1996.

[KS99] Gabor Karsai and Janos Sztipanovits. A Model-Based Approach to Self-Adaptive Software. *IEEE Intelligent Systems*, 14(3):46–53, 1999.

[KSF02] Tore Knabe, Michael Schillo, and Klaus Fischer. Improvements to the FIPA Contract Net Protocol for Performance Increase and Cascading Applications. In *Proceedings of the Workshop on Multiagent Interoperability (MAI 2002)*, 2002.

[KSGM03] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The EigenTrust Algorithm for Reputation Management in P2P Networks. In *Proceedings of the 12th International Conference on World Wide Web (WWW 2003)*, pages 640–651. ACM, 2003.

[KT86] Peter N. Kugler and Michael T. Turvey. *Information, Natural Law, and the Self-Assembly of Rhythmic Movement*. Lawrence Erlbaum, 1986.

[Lan86] Christopher Gale Langton. Studying artificial life with cellular automata. *Physica D*, 2(1-3):120–149, 1986.

[Lar01] Allan Larsen. *The Dynamic Vehicle Routing Problem*. Phd thesis, Department of Mathematical Modelling, Technical University of Denmark, 2001.

[LAW02] Chenyang Lu, Guillermo A. Alvarez, and John Wilkes. Aqueduct: Online Data Migration with Performance Guarantees. In *Proceedings of the 1st USENIX Conference on File and Storage Technologies (FAST 2002)*, pages 219–230. USENIX Association, 2002.

[Len64] George G. Lendaris. On the definition of self-organizing systems. *Proceedings of the IEEE*, 52(3):324–325, March 1964.

[Lew75] George Henry Lewes. *Problems of life and mind*, volume 2. Kegan Paul, Trench, Turbner, London, 1875.

[Lew84] Trevor Lewis. The elements and frontiers of insect communication. In Trevor Lewis, editor, *Insect Communication*, pages 1–27. Academic Press, London, 1984.

[LHH+02] Zhijian Lu, Jason Hein, Marty Humphrey, Mircea Stan, John Lach, and Kevin Skadron. Control-theoretic dynamic frequency and voltage scaling for multimedia workloads. In *Proceedings of the 2002 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES 2002)*, pages 156–163. ACM, 2002.

[Lid79] William Z. Lidicker, Jr. A Clarification of Interactions in Ecological Systems. *BioScience*, 29(8):475–477, Aug 1979.

[Lin03] Jürgen Lind. Patterns in Agent-Oriented Software Engineering. In Fausto Giunchiglia, James Odell, and Gerhard Weiß, editors, *Proceedings of the 3rd International Workshop on Agent-Oriented Software Engineering III (AOSE 2002), Revised Papers and Invited Contributions*, volume 2585 of *Lecture Notes in Computer Science*, pages 47–58. Springer, 2003.

[LL01] Haibing Li and Andrew Lim. A Metaheuristic for the Pickup and Delivery Problem with Time Windows. In *Proceedings of the 13th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2001)*, pages 160–170, 2001.

[LL02] Haibing Li and Andrew Lim. Local search with annealing-like restarts to solve the vehicle routing problem with time windows. In *Proceedings of the 2002 ACM Symposium on Applied Computing (SAC 2002)*, pages 560–565. ACM Press, 2002.

[LL06] Hon Wai Leong and Ming Liu. A multi-agent algorithm for vehicle routing problem with time window. In *Proceedings of the 2006 ACM Symposium on Applied Computing (SAC 2006)*, pages 106–111. ACM, 2006.

[LMK+05] Michail G. Lagoudakis, Evangelos Markakis, David Kemp, Pinar Keskinocak, Anton Kleywegt, Sven Koenig, Craig Tovey, Adam Meyerson, and Sonal Jain. Auction-Based Multi-Robot Routing. In *Proceedings of the International Conference on Robotics: Science and Systems (ROBOTICS 2005)*, pages 343–350. Springer, 2005.

[LMR96] Karsten Lund, Oli B. G. Madsen, and Jens M. Rygaard. Vehicle Routing Problems with Varying Degrees of Dynamism. Technical report, Department of Mathematical Modelling, Technical University of Denmark, 1996.

[LMSK63] John D. C. Little, Katta G. Murty, Dura W. Sweeney, and Caroline Karel. An Algorithm for the Traveling Salesman Problem. *Operations Research*, 11(6):972–989, 1963.

[LO98] Danny B. Lange and Mitsuru Oshima. Mobile agents with Java: The Aglet API. *World Wide Web*, 1(3):111–121, 1998.

[LR71] John H. Law and Fred E. Regnier. Pheromones. *Annual Review of Biochemistry*, 40:533–548, 1971.

[LRS03a] Robert Laddaga, Paul Robertson, and Howard E. Shrobe. Introduction to Self-adaptive Software: Applications. In *Proceedings of the 2nd International Workshop Self-Adaptive Software (IWSAS 2001), Revised Papers*, volume 2614 of *Lecture Notes in Computer Science*, pages 1–5. Springer, 2003.

[LRS03b] Robert Laddaga, Paul Robertson, and Howard E. Shrobe, editors. *Proceedings of the 2nd International Workshop on Self-Adaptive Software, Revised Papers (IWSAS 2001)*, volume 2614 of *Lecture Notes in Computer Science*. Springer, 2003.

[LSST02] Chenyang Lu, John A. Stankovic, Sang Hyuk Son, and Gang Tao. Feedback Control Real-Time Scheduling: Framework, Modeling, and Algorithms. *Real-Time Systems*, 23(1–2):85–126, 2002.

[LWL07] Henry Y.K. Lau, Vicky W.K. Wong, and Ivan S.K. Lee. Immunity-based autonomous guided vehicles control. *Applied Soft Computing*, 7(1):41–57, 2007.

[Mad08] MadKit. http://www.madkit.org, 2008.

[Mas99] Saverio Mascolo. Classical Control Theory for Congestion avoidance in High-Speed Internet. In *Proceedings of the 38th IEEE Conference on Decision and Control (CDC 1999)*, pages 2709–2714. IEEE Computer Society, 1999.

[MAS08] MASON. http://www.cs.gmu.edu/~eclab/projects/mason, 2008.

[Max68] James C. Maxwell. On Governors. *Proceedings of the Royal Society of London*, 16:270–283, 1868.

[MB07] Daniel A. Menascé and Mohamed N. Bennani. Dynamic Server Allocation for Autonomic Service Centers in the Presence of Failures. In Manish Parashar and Salim Hariri, editors, *Autonomic Computing: Concepts, Infrastructure, and Applications*, pages 353–368. CRC Press, 2007.

[MBBY06] Ali A. Minai, Dan Braha, and Yaneer Bar-Yam. Complex Engineered Systems: A New Paradigm. In *Complex Engineered Systems*, pages 1–21. Springer, 2006.

[MC94] Thomas W. Malone and Kevin Crowston. The Interdisciplinary Study of Coordination. *ACM Computing Surveys*, 26(1):87–199, 1994.

[MCE01] Cecilia Mascolo, Licia Capra, and Wolfgang Emmerich. An XML-based Middleware for Peer-to-Peer computing. In *Proceedings of the 1st International Conference on Peer-to-Peer Computing (P2P 2001)*, page 69. IEEE Computer Society, 2001.

[MCR06] Justin Moore, Jeffrey S. Chase, and Parthasarathy Ranganathan. Weatherman: Automated, Online and Predictive Thermal Mapping and Management for Data Centers. In *Proceedings of the 3rd IEEE International Conference on Autonomic Computing 2006 (ICAC 2006)*, pages 155–164. IEEE, 2006.

[MD97] Gerard Meszaros and Jim Doble. A pattern language for pattern writing. In *Pattern languages of program design 3*, pages 529–574. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[MDK93] Jeff Magee, Naranker Dulay, and Jeff Kramer. Structuring parallel and distributed programs. *Software Engineering Journal*, 8(2):73–82, 1993.

[MFSG05] Jean-Philippe Martin-Flatin, Joe Sventek, and Kurt Geihs, editors. *Proceedings of the 1st IFIP/IEEE International Workshop on Self-Managed Systems and Services (SelfMan 2005)*. IEEE Computer Society, 2005.

[MGRD05] Roberto Montemanni, Luca M. Gambardella, Andrea E. Rizzoli, and Alberto V. Donati. Ant Colony System for a Dynamic Vehicle Routing Problem. *Journal of Combinatorial Optimization*, 10(4):327–343, 2005.

[Müh02] Gero Mühl. *Large-Scale Content-Based Publish/Subscribe Systems*. Phd thesis, University of Darmstadt, 2002.

[Mic03] Microsoft Corporation. Dynamic Systems Initiative. `http://www.microsoft.com/business/dsi/default.mspx`, 2003.

[Mic07] Microsoft Corporation. Dynamic Systems Initiative Whitepaper, 2007.

[Mic08] Microsoft Corporation. Microsoft Encarta 2008, 2008.

[Min22] Nicolas Minorsky. Directional Stability and Automatically Steered Bodies. *Journal of the American Society for Naval Engineers*, 34(2):280–309, 1922.

[ML95] Naftaly H. Minsky and Jerrold Leichter. Law-Governed Linda as a Coordination Model. In *Proceedings of the Workshop on Models and Languages for Coordination of Parallelism and Distribution, Object-Based Models and Languages for Concurrent Systems (ECOOP 1994), Selected Papers*, volume 924 of *Lecture Notes in Computer Science*, pages 125–146. Springer, 1995.

[Mül96] Jörg P. Müller. *The Design of Intelligent Agents: A Layered Approach.* Number 1177 in Lecture Notes of Artificial Intelligence. Springer, 1996.

[MMKL04] Snežana Mitrović-Minić, Ramesh Krishnamurti, and Gilbert Laporte. Double-horizon based heuristics for the dynamic pickup and delivery problem with time windows. *Transportation Research Part B: Methodological*, 38(8):669–685, 2004.

[MML04] Snežana Mitrović-Minić and Gilbert Laporte. Waiting strategies for the dynamic pickup and delivery problem with time windows. *Transportation Research Part B: Methodological*, 38(7):635–655, 2004.

[MML09] Alejandro R. Mosteo, Luis Montano, and Michail G. Lagoudakis. Guaranteed-Performance Multi-robot Routing under Limited Communication Range. In *Distributed Autonomous Robotic Systems*, volume 8. Springer, 2009.

[MMS06] Moez Mnif and Christian Müller-Schloer. Quantitative Emergence. In *Proceedings of the 2006 IEEE Mountain Workshop on Adaptive and Learning Systems*, pages 78–84. IEEE, 2006.

[MMTZ06] Marco Mamei, Ronaldo Menezes, Robert Tolksdorf, and Franco Zambonelli. Case studies for Self-organization in Computer Science. *Journal of Systems Architecture: the EUROMICRO Journal*, 52(8):443–460, 2006.

[Mog06] Jeffrey C. Mogul. Emergent (mis)behavior vs. complex software systems. *ACM SIGOPS Operating Systems Review*, 40(4):293–304, 2006.

[Moo65] Gordon E. Moore. Cramming More Components onto Integrated Circuits. *Electronics*, 38(8):114–117, 1965.

[Mor26] Conwy Lloyd Morgan. *Emergent evolution: The Gifford lectures, delivered in the University of St. Andrews in the year 1922.* Henry Holt, 1926.

[MPR06] Amy L. Murphy, Gian Pietro Picco, and Gruia-Catalin Roman. LIME: A coordination model and middleware supporting mobility of hosts and agents. *ACM Transactions on Software Engineering and Methodology*, 15(3):279–328, 2006.

[MRB+07] Moez Mnif, Urban Richter, Jürgen Branke, Hartmut Schmeck, and Christian Müller-Schloer. Measurement and Control of Self-organised Behaviour in Robot Swarms. In *Proceedings of the 20th International Conference on Architecture of Computing Systems (ARCS 2007)*, volume 4415 of *Lecture Notes in Computer Science*, pages 209–223. Sp, 2007.

[MRF+03] Soraya Kouadri Mostéfaoui, Omer F. Rana, Noria Foukia, Salima Hassas, Giovanna Di Marzo Serugendo, Chris Van Aart, and Anthony Karageorgos. Self-Organzing Applications: A Survey. In Giovanna Di Marzo Serugendo, Anthony Karageorgos, Omer F. Rana, and Franco Zambonellini, editors, *Proceedings of the International Workshop on Engineering Self-Organising Applications (ESOA 2003)*, pages 62–69, 2003.

[MS08] Hausi Müller and Mauro Pezzèand Mary Shaw. Visibility of control in adaptive systems. In *Proceedings of the 2nd International Workshop on Ultra-Large-Scale Software-Intensive Systems (ULSSIS 2008)*, pages 23–26. ACM, 2008.

[MSdWdW08] Tamas Máhr, Jordan Srour, Mathijs de Weerdt, and Mathijs de Weerdt. Agent Performance in Vehicle Routing when the Only Thing Certain is Uncertainty. In *Proceedings of 7th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008), Workshop on Agents in Traffic and Transportation (ATT 2008)*, 2008.

[MSdWZ10] Tamas Mahr, Jordan Srour, Mathijs M. de Weerdt, and Rob Zuidwijk. Can agents measure up? A comparative study of an agent-based and on-line optimization approach for a drayage problem with uncertainty. *Transportation Research: Part C*, 18(1):99–119, 2010.

[MSKC04] Philip K. McKinley, Seyed Masoud Sadjadi, Eric P. Kasten, and Betty H. C. Cheng. Composing Adaptive Software. *Computer*, 37(7):56–64, 2004.

[MSM+09] Alberto Montresor, Fabrice Saffre, Nenad Medvidovic, Jake Beal, and Salima Hassas, editors. *Proceedings of the 3rd IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2009)*. IEEE Computer Society, 2009.

[MSvdMW04] Christian Müller-Schloer, Christoph von der Malsburg, and Rolf P. Würtz. Organic Computing. *Informatik Spektrum*, 27(5):332–336, 2004. in German.

[MT03] Ronaldo Menezes and Robert Tolksdorf. A new approach to scalable Linda-systems based on swarms. In *Proceedings of the 18th ACM Symposium on Applied Computing (SAC 2003)*, pages 375–379, 2003.

[Mur04] Richard Murch. *Autonomic Computing*. IBM Press, 2004.

[MvdHvH08] Martijn Mes, Matthieu van der Heijden, and Jos van Hillegersberg. Design choices for agent-based control of AGVs in the dough making process. *Decision Support Systems*, 44(4):983–999, 2008.

[Mvv07] Martijn Mes, Matthieu van der Heijden, and Aart van Harten. Comparison of agent-based scheduling to look-ahead heuristics for real-time transportation problems. *European Journal of Operational Research*, 181(1):59–75, 2007.

[MWJ+07] Gero Mühl, Matthias Werner, Michael A. Jaeger, Klaus Herrmann, and Helge Parzyjegla. On the Definitions of Self-Managing and Self-Organizing Systems. In T. Braun, G. Carle, and B. Stiller, editors, *Proceedings of the KiVS 2007 Workshop: Selbstorganisierende, Adaptive, Kontextsensitive verteilte Systeme (SAKS 2007)*, pages 291–301. VDE Verlag, 2007.

[MZ04] Marco Mamei and Franco Zambonelli. Co-Fields: A Physically Inspired Approach to Motion Coordination. *IEEE Pervasive Computing*, 3(2):52–61, 2004.

[MZ06] Marco Mamei and Franco Zambonelli. *Field-Based Coordination for Pervasive Multiagent Systems*. Springer Series on Agent Technology. Springer-Verlag, 2006.

[MZ09] Marco Mamei and Franco Zambonelli. Programming pervasive and mobile computing applications: the tota approach. *ACM Transaction on Software Engineering and Methodology*, 18(4):1–1, 2009.

[MZL03] Marco Mamei, Franco Zambonelli, and Letizia Leonardi. Tuples On The Air: a Middleware for Context-Aware Computing in Dynamic Networks. In *Proceedings of the 23rd International Conference on Distributed Computing Systems Workshops (ICDCSW 2003)*, pages 342–347. IEEE Computer Society, 2003.

[Net08] NetLogo. http://ccl.northwestern.edu/netlogo, 2008.

[New96] David V. Newman. Emergence and Strange Attractors. *Philosophy of Science*, 63(2):245–261, 1996.

[Nis07] Norman S. Nise. *Control Systems Engineering*. John Wiley & Sons, 2007.

[NL76] Donald A. Nordlund and W. J. Lewis. Terminology of chemical releasing stimuli in intraspecific and interspecific interactions. *Journal of Chemical Ecology*, 2(2):211–220, April 1976.

[NLJ97] Hyacinth S. Nwana, Lyndon C. Lee, and Nicholas R. Jennings. Coordination in multi-agent systems. In Hyacinth S. Nwana and Nader Azarmi, editors, *Software Agents and Soft Computing: Towards Enhancing Machine Intelligence, Concepts and Applications*, volume 1198 of *Lecture Notes in Computer Science*, pages 42–58. Springer, 1997.

[Nor81] Donald A. Nordlund. Semiochemicals: A Review of the Terminology. In Donald A. Nordlund, Richard L. Jones, and W. Joe Lewis, editors, *Semiochemicals: Their Role in Pest Control*, pages 13–28. John Wiley & Sons, New York, 1981.

[NOR03] Donald A. Norman, Andrew Ortony, and Daniel M. Russell. Affect and machine design: Lessons for the development of autonomous machines. *IBM Systems Journal*, 42(1):38–44, 2003.

[NP77] Grégoire Nicolis and Ilya Prigogine. *Self-Organization in Non-Equilibrium Systems: From Dissipative Structures to Order Through Fluctuations*. Wiley, New York, 1977.

[NR01] Emma Norling and Frank E. Ritter. Embodying the JACK Agent Architecture. In *Proceedings of the 14th Australian Joint Conference on Artificial Intelligence (AI 2001)*, volume 2256 of *Lecture Notes in Computer Science*, pages 368–377. Springer-Verlag, 2001.

[NT04] Sunil Nakrani and Craig Tovey. On Honey Bees and Dynamic Server Allocation in Internet Hosting Centers. *Adaptive Behavior - Animals, Animats, Software Agents, Robots, Adaptive Systems*, 12(3-4):223–240, 2004.

[Nun06] Leandro Nunes de Castro. *Fundamentals of Natural Computing*. Chapman & Hall/CRC, 2006.

[Nyq32] Harry Nyquist. Regeneration Theory. *Bell System Technical Journal*, 11:126–147, 1932.

[O'C94] Timothy O'Connor. Emergent Properties. *American Philosophical Quarterly*, 31:91–104, 1994.

[OCI05] OCI. Organic Computing Initiative. http://www.organic-computing.de, 2005.

[ODN95] Andrea Omicini, Enrico Denti, and Antonio Natali. Agent Coordination and Control through Logic Theories. In *Proceedings of the 4th Congress of the Italian Association for Artificial Intelligence on Topics in Artificial Intelligence (AI*IA 1995)*, volume 992 of *Lecture Notes in Computer Science*, pages 439–450. Springer, 1995.

[Oga09] Katsuhiko Ogata. *Modern Control Engineering*. Prentice Hall, 5th edition, 2009.

[OGT+99] Peyman Oreizy, Michael M. Gorlick, Richard N. Taylor, Dennis Heimbigner, Gregory Johnson, Nenad Medvidovic, Alex Quilici, David S. Rosenblum, and Alexander L. Wolf. An Architecture-Based Approach to Self-Adaptive Software. *IEEE Intelligent Systems*, 14(3):54–62, 1999.

[OM06] Sascha Ossowski and Ronaldo Menezes. On coordination and its significance to distributed and multi-agent systems. *Concurrency and Computation: Practice and Experience*, 18(4):359–370, 2006.

[OMG04a] OMG. CORBA Event Service Specification, Version 1.2. http://www.omg.org/docs/formal/04-10-02.pdf, October 2004.

[OMG04b] OMG. CORBA Notification Service Specification, Version 1.1. http://www.omg.org/docs/formal/04-10-11.pdf, October 2004.

[OMG07] OMG. Data Distribution Service for Real-time Systems, Version 1.2. http://www.omg.org/docs/formal/07-01-01.pdf, January 2007.

[Omi01] Andrea Omicini. SODA: societies and infrastructures in the analysis and design of agent-based systems. In *Proceedings of the 1st International Workshop on Agent-oriented Software Engineering (AOSE 2000)*, pages 185–193. Springer-Verlag, 2001.

[OMT08] Peyman Oreizy, Nenad Medvidovic, and Richard N. Taylor. Runtime software adaptation: framework, approaches, and styles. In *Companion of the 30th International Conference on Software Engineering (ICSE Companion 2008)*, pages 899–910. ACM, 2008.

[OO03] Andrea Omicini and Sascha Ossowski. Objective versus Subjective Coordination in the Engineering of Agent Systems. In *Intelligent Information Agents - The AgentLink Perspective*, volume 2586 of *Lecture Notes in Computer Science*, pages 179–202. Springer, 2003.

[OOR04] Andrea Omicini, Sascha Ossowski, and Alessandro Ricci. Coordination Infrastructures in the Engineering of Multiagent Systems. In

Federico Bergenti, Marie-Pierre Gleizes, and Franco Zambonelli, editors, *Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook*, Multiagent Systems, Artificial Societies, and Simulated Organizations, chapter 14, pages 273–296. Kluwer Academic Publishers, 2004.

[OPBS04] James J. Odell, H. Van Dyke Parunak, Sven Brueckner, and John Sauter. Temporal Aspects of Dynamic Role Assignment. In *Proceedings of the 4th International Workshop on Agent-Oriented Software Engineering (AOSE 2003), Revised Papers*, volume 2935 of *Lecture Notes in Computer Science*, pages 185–214. Springer, 2004.

[ORA09] ORACLE. Oracle Application Grid. http://www.oracle.com/products/middleware/application-grid.html, 2009.

[ORV08] Andrea Omicini, Alessandro Ricci, and Mirko Viroli. Artifacts in the A&A meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 17(3):432–456, 2008.

[ORVR04] Andrea Omicini, Alessandro Ricci, Mirko Viroli, and Giovanni Rimassa. Integrating objective & subjective coordination in multi-agent systems. In *Proceedings of the 2004 ACM Symposium on Applied Computing (SAC 2004)*, pages 449–455. ACM, 2004.

[Oss99] Sascha Ossowski. *Co-ordination in Artificial Agent Societies*, volume 1535 of *Lecture Notes in Computer Science*. Springer, 1999.

[OZ98] Andrea Omicini and Franco Zambonelli. TuCSoN: a Coordination model for Mobile Information Agents. In *Proceedings of the 1st International Workshop on Innovative Internet Information Systems (IIIS 1998)*, pages 177–187. IDI – NTNU, Trondheim, 1998.

[OZ05] Stephan Olariu and Albert Y. Zomaya, editors. *Handbook of Bioinspired Algorithms andAapplications*. CRC Computer and Information Science. Chapman & Hall, 2005.

[PA97] George A. Papadopoulos and Farhad Arbab. Coordination of Distributed and Parallel Activities in the IWIM Model. *International Journal of High Speed Computing*, 9(2):127–160, 1997.

[PA98] George A. Papadopoulos and Farhad Arbab. Coordination Models and Languages. *Advances in Computers*, 46:329–400, 1998.

[Pag88] Heinz Pagels. *The Dreams of Reason: The Computer and the Rise of the Sciences of Complexity*. Simon and Schuster, New York, 1988.

[Pan05] Giselher Pankratz. Dynamic vehicle routing by means of a genetic algorithm. *International Journal of Physical Distribution & Logistics Management*, 35(5):362–383, 2005.

[Pap01] George A. Papadopoulos. Models and Technologies for the Coordination of Internet Agents: A Survey. In *Coordination of Internet agents: models, technologies, and applications*, pages 25–56. Springer-Verlag, Lodon, UK, 2001.

[Par82] Brian L. Partridge. The structure and function of fish schools. *Scientific American*, 246(6):114–123, 1982.

[PB01] H. Van Dyke Parunak and Sven Brückner. Entropy and Self-Organization in Multi-Agent Systems. In Jörg P. Müller, Elisabeth André, Sandip Sen, and Claude Frasson, editors, *Proceedings of the 5th International Conference on Autonomous Agents (AGENTS 2001)*, pages 124–130. ACM Press, 2001.

[PB02] Peter R. Pietzuch and Jean Bacon. Hermes: A Distributed Event-Based Middleware Architecture. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCSW 2002)*, pages 611–618. IEEE Computer Society, 2002.

[PB04] H. Van Dyke Parunak and Sven A. Brueckner. Engineering Swarming Systems. In Federico Bergenti, Marie-Pierre Gleizes, and Franco Zambonelli, editors, *Methodologies and Software Engineering for Agent Systems*, volume 11 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*. Springer US, 2004.

[PBS04] H. Van Dyke Parunak, Sven Brückner, and John A. Sauter. Digital Pheromones for Coordination of Unmanned Vehicles. In *Proceedings of the 1st International Workshop on Environments for Multi-Agent Systems (E4MAS 2004), Revised Selected Papers*, volume 3374 of *Lecture Notes in Computer Science*, pages 246–263. Springer, 2004.

[PDH08a] Sophie N. Parragh, Karl F. Doerner, and Richard F. Hartl. A survey on pickup and delivery problems - Part I: Transportation between customers and depot. *Journal für Betriebswirtschaft*, 58(1):21–51, April 2008.

[PDH08b] Sophie N. Parragh, Karl F. Doerner, and Richard F. Hartl. A survey on pickup and delivery problems - Part II: Transportation between pickup and delivery locations. *Journal für Betriebswirtschaft*, 58(2):81–117, June 2008.

[Pet98] Peter Wyckoff and Stephen W. McLaughry and Tobin J. Lehman and Daniel Alexander Ford. T spaces. *IBM Systems Journal*, 37(3):454–474, 1998.

[PF03] Shrideep Pallickara and Geoffrey Fox. NaradaBrokering: A Distributed Middleware Framework and Architecture for Enabling

Durable Peer-to-Peer Grids. In *Proceedings of the ACM/IFIP/USENIX International Middleware Conference (Middleware 2003)*, pages 41–61, 2003.

[PFJ+01] João Pereira, Françoise Fabret, Hans-Arno Jacobsen, Françoise Llirbat, and Dennis Shasha. WebFilter: A High-throughput XML-based Publish and Subscribe System. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB 2001)*, pages 723–724. Morgan Kaufmann Publishers Inc., 2001.

[PFL+00] João Pereira, Françoise Fabret, Françoise Llirbat, Radu Preotiuc-Pietro, Kenneth A. Ross, and Dennis Shasha. Publish/Subscribe on the Web at Extreme Speed. In *Proceedings of the 26th International Conference on Very Large Data Bases (VLDB 2000)*, pages 627–630. Morgan Kaufmann Publishers Inc., 2000.

[PGH+02] Sujay Parekh, Neha Gandhi, Joe Hellerstein, Dawn Tilbury, T. S. Jayram, and Joe Bigus. Using Control Theory to Achieve Service Level Objectives in Performance Management. *Real-Time Systems*, 23(1–2):127–141, 2002.

[PGK+06] Duc-Truong Pham, Afshin Ghanbarzadeh, Ebubekir Koç, Sameh Otri, S. Rahim, and Muhamad Zaidi. The Bees Algorithm – A Novel Tool for Complex Optimisation Problems. In *Proceedings of the 2nd Virtual International Conference on Intelligent Production Machines and Systems (IPROMS 2006)*, pages 454–459, 2006.

[PGSFF05] Juan Pavón, Jorge J. Gómez-Sanz, and Rubén Fuentes-Fernández. The INGENIAS Methodology and Tools. In Brian Henderson-Sellers and Paolo Giorgini, editors, *Agent-Oriented Methodologies*, pages 236–276. Idea Group Publishing, 2005.

[PH05] Manish Parashar and Salim Hariri. Autonomic Computing: An Overview. In Jean-Pierre Banâtre, Pascal Fradet, Jean-Louis Giavitto, and Olivier Michel, editors, *Proceedings of the International Workshop on Unconventional Programming Paradigms (UPP 2004), Revised Selected and Invited Papers*, volume 3566 of *Lecture Notes in Computer Science*, pages 257–269. Springer, 2005.

[Pin08] Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer, 3rd edition, 2008.

[PJO95] Warren B. Powell, Patrcik Jaillet, and Amedeo Odoni. Stochastic and Dynamic Networks and Routing. In M.O. Ball, T.L. Magnanti, C.L. Monma, and G.L. Nemhauser, editors, *Handbooks in Operations Research and Management Science: Network Routing*, volume 8, pages 141–295. North-Holland, Amsterdam, 1995.

[PL04] Liviu Panait and Sean Luke. A Pheromone-Based Utility Model for Collaborative Foraging. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems 8AAMAS 2004)*, pages 36–43. IEEE Computer Society, 2004.

[Plo64] Edward Grosvenor Plowman. *Elements of Business Logistics*. Stanford University Press, Stanford, 1964.

[PLSP03] Don Perugini, Dale Lambert, Leon Sterling, and Adrian Pearce. A Distributed Agent Approach to Global Transportation Scheduling. In *Proceedings of the IEEE/WIC International Conference on Intelligent Agent Technology (IAT 2003)*, page 18. IEEE Computer Society, 2003.

[PMN05] Aris A. Papadopoulos, Julie A. McCann, and Alfredo Navarra. Connectionless Probabilistic (CoP) routing: an efficient protocol for mobile wireless ad-hoc sensor networks. In *Proceedings of 24th IEEE International Performance, Computing, and Communications Conference (IPCCC 2005)*, pages 73–77. IEEE Computer Society, 2005.

[PMS+07] Eric Platon, Marco Mamei, Nicolas Sabouret, Shinichi Honiden, and H. Van Dyke Parunak. Mechanisms for environments in multi-agent systems: Survey and opportunities. *Autonomous Agents and Multi-Agent Systems*, 14(1):31–47, 2007.

[Pop06] Douglas A. Popken. Controlling order circuity in pickup and delivery problems. *Transportation Research Part E: Logistics and Transportation Review*, 42(5):431–443, 2006.

[PRT+08] Holger Prothmann, Fabian Rochner, Sven Tomforde, Jürgen Branke, Christian Müller-Schloer, and Hartmut Schmeck. Organic Control of Traffic Lights. In *Proceedings of the 5th International Conference on Autonomic and Trusted Computing (ATC 2008)*, volume 5060 of *Lecture Notes in Computer Science*, pages 219–233. Springer, 2008.

[PS97] Holger Peine and Torsten Stolpmann. The Architecture of the Ara Platform for Mobile Agents. In *Proceedings of the 1st International Workshop on Mobile Agents (MA 1997)*, volume 1219 of *Lecture Notes In Computer Science*, pages 50–61. Springer, 1997.

[Psa88] Harilaos N. Psaraftis. Dynamic Vehicle Routing Problems. In Bruce Golden and A. Assad, editors, *Vehicle Routing: Methods and Studies*, pages 223–248. North-Holland, 1988.

[Psa95] Harilaos N. Psaraftis. Dynamic vehicle routing: Status and prospects. *Annals of Operations Research*, 61(1):143–164, 1995.

[PSD95] Jean-Yves Potvin, Yu Shen, and Gina Dufour. Learning techniques for an expert vehicle dispatching system. *Expert Systems with Applications*, 8(1):101–109, 1995.

[PSH07] Eric Platon, Nicolas Sabouret, and Shinichi Honiden. Environmental Support for Tag Interactions. In *Proceedings of the 3rd International Workshop on Environments for Multi-Agent Systems (E4MAS 2006), Selected Revised and Invited Papers*, volume 4389 of *Lecture Notes in Computer Science*, pages 106–123. Springer, 2007.

[PT09] Francisco Baptista Pereira and Jorge Tavares, editors. *Bio-inspired Algorithms for the Vehicle Routing Problem*, volume 161 of *Studies in Computational Intelligence*. Springer, 2009.

[PW04] Lin Padgham and Michael Winikoff. *Developing Intelligent Agent Systems: A Practical Guide*. John Wiley & Sons, 2004.

[PYL96] Michael Proctor, Peter Yeo, and Andrew Lack. *The Natural History of Pollination*. Timber Press, 1996.

[RC90] Gruia-Catalin Roman and H. Conrad Cunningham. Mixed Programming Metaphors in a Shared Dataspace Model of Concurrency. *IEEE Transactions on Software Engineering*, 16(12):1361–1373, 1990.

[RCB06] Alan Rushton, Phil Croucher, and Peter Baker. *The Handbook of Logistics and Distribution Management*. Kogan Page, 2006.

[Rep08] Repast. Recursive Porous Agent Simulation Toolkit. `http://repast.sourceforge.net`, 2008.

[Res94] Mitchel Resnick. *Turtles, Termites and Traffic Jams: Explorations in Massively Parallel Microworlds*. Complex Adaptive Systems. MIT Press, Cambridge, Massachusetts, 1994.

[RG95] Anand S. Rao and Michael P. Georgeff. BDI Agents: From Theory to Practice. In *Proceedings of the 1st International Conference on Multi-Agent Systems (ICMAS 1995)*, pages 312–319, 1995.

[RGS+05] Joe R. Riley, Uwe Greggers, Alan D. Smith, Don R. Reynolds, and Randolf Menzel. The flight paths of honeybees recruited by the waggle dance. *Nature*, 435:205–207, 2005.

[RIF02] Matei Ripeanu, Adriana Iamnitchi, and Ian Foster. Mapping the Gnutella Network. *IEEE Internet Computing*, 6(1):50–57, 2002.

[RL68] Fred E. Regnier and John H. Law. Insect pheromones. *Journal of Lipid Research*, 9:541–551, 1968.

[RMB+06] Urban Richter, Moez Mnif, Jürgen Branke, Christian Müller-Schloer, and Hartmut Schmeck. Towards a generic observer/controller architecture for Organic Computing. In Christian Hochberger and Rüdiger Liskowsky, editors, *Informatik 2006 - Informatik für Menschen*, volume 93 of *Lecture Notes in Informatics*, pages 112–119. GI, 2006.

[RMP+07] Ian Rose, Rohan Murty, Peter Pietzuch, Jonathan Ledlie, Mema Roussopoulos, and Matt Welsh. Cobra: Content-based Filtering and Aggregation of Blogs and RSS Feeds. In *Proceedings of the 4th USENIX Symposium on Networked Systems Design & Implementation (NSDI 2007)*, 2007.

[RMS02] Joachim Ruther, Torsten Meiners, and Johannes L. M. Steidle. Rich in phenomena-lacking in terms. A classification of kairomones. *Chemoecology*, 12(4):161–167, November 2002.

[RN02] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd edition, 2002.

[RP06] Stefan Ropke and David Pisinger. An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows. *Transportation Science*, 40(4):455–472, 2006.

[RRS08] Oliver Ribock, Urban Richter, and Hartmut Schmeck. Using Organic Computing to Control Bunching Effects. In *Proceedings of the 21st International Conference on Architecture of Computing Systems (ARCS 2008)*, volume 4934 of *Lecture Notes in Computer Science*, pages 232–244. Springer, 2008.

[RS06] Wolfgang Renz and Jan Sudeikat. Mesoscopic Modeling of Emergent Behavior - A Self-organizing Deliberative Minority Game. In Sven Brueckner, Giovanna Di Marzo Serugendo, David Hales, and Franco Zambonelli, editors, *Proceedings of the 3rd International Workshop on Engineering Self-Organising Systems (ESOA 2005) Revised Selected Papers*, volume 3910 of *Lecture Notes in Computer Science*, pages 167–181. Springer, 2006.

[RSC99] Edmund M. A. Ronald, Moshe Sipper, and Mathieu S. Capcarrère. Testing for Emergence in Artificial Life. In *Proceedings of the 5th European Conference on Advances in Artificial Life (ECAL 1999)*, volume 1674 of *Lecture Notes in Computer Science*, pages 13–20. Springer, 1999.

[RSL01] Paul Robertson, Howard E. Shrobe, and Robert Laddaga, editors. *Proceedings of the 1st International Workshop on Self-Adaptive Soft-*

ware, Revised Papers (IWSAS 2000), volume 1936 of *Lecture Notes in Computer Science*. Springer, 2001.

[RW97] Antony I. T. Rowstron and Alan M. Wood. Bonita: A set of tuple space primitives for distributed coordination. In *Proceedings of the 30th Hawaii International Conference on System Sciences (HICSS 1997)*, page 379. IEEE Computer Society, 1997.

[RY94] K. K. Ramakrishnan and Henry Yang. The Ethernet capture effect: analysis and solution. In *Proceedings of the 19th Conference on Local Computer Networks (LCN 1994)*, pages 228–240. IEEE Computer Society, 1994.

[SA97] Bill Segall and David Arnold. Elvin has left the building: A publish/subscribe notification service with quenching. In *Proceedings of the 1997 Australian UNIX and Open Systems Users Group Conference (AUUG 1997)*, 1997.

[Sar93] Vijay A. Saraswat. *Concurrent Constraint Programming.* MIT Press, 1993.

[SARDS01] Alberto Rodrigues Silva, ao Artur Rom Dwight Deugo, and Miguel Mira Da Silva. Towards a Reference Model for Surveying Mobile Agent Systems. *Autonomous Agents and Multi-Agent Systems*, 4(3):187–231, 2001.

[Saw03] Keith R. Sawyer. Artificial Societies: Multiagent Systems and the Micro-Macro Link in Sociological Theory. *Sociological Methods & Research*, 31(3):325–363, 2003.

[SBC+98] Robert E. Strom, Guruduth Banavar, Tushar Deepak Chandra, Marc Kaplan, Kevan Miller, Bodhi Mukherjee, Daniel C. Sturman, and Michael Ward. Gryphon: An Information Flow Based Approach to Message Brokering. In *Proceedings of the International Symposium on Software Reliability Engineering (ISSRE1998)*, 1998.

[SBC05] Thirunavukkarasu Sivaharan, Gordon Blair, and Geoff Coulson. GREEN: A Configurable and Re-configurable Publish-Subscribe Middleware for Pervasive Computing. In *Proceedings of the OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2005*, volume 3760 of *Lecture Notes in Computer Science*, pages 732–749. Springer, 2005.

[SBG04] Tajana Simunic, Stephen P. Boyd, and Peter Glynn. Managing Power Consumption in Networks on Chips. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 12(1):96–107, 2004.

[SBH+06] Stephen Strowes, Nagwa Badr, Steven Heeps, Emil Lupu, and Morris Sloman. An Event Service Supporting Autonomic Management of Ubiquitous Systems for e-Health. In *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW 2006)*, pages 22–27. IEEE Computer Society, 2006.

[Sch78] Thomas C. Schelling. *Micromotives and Macrobehavior*. W. W. Norton, New York, 1978.

[Sch97] Frank Schweitzer, editor. *Self-Organization of Complex Structures: From Individual to Collective Dynamics*. Gordon and Breach, 1997.

[SCH99] Michael Schumacher, Fabrice Chantemargue, and Béat Hirsbrunner. The STL++ Coordination Language: A Base for Implementing Distributed Multi-agent Applications. In *Proceedings of the 3rd International Conference on Coordination Languages and Models (COORDINATION 1999)*, volume 1594 of *Lecture Notes In Computer Science*, pages 399–414. Springer, 1999.

[Sch01] Michael Schumacher. *Objective coordination in multi-agent system engineering: design and implementation*, volume 2039 of *Lecture Notes in Artificial Intelligence*. Springer, 2001.

[Sch05] Hartmut Schmeck. Organic Computing - A New Vision for Distributed Embedded Systems. In *Proceedings of the 8th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2005)*, pages 201–203. IEEE Computer Society, 2005.

[SCN08] Doris Sáez, Cristián E. Cortés, and Alfredo Núñez. Hybrid adaptive predictive control for the multi-vehicle dynamic pick-up and delivery problem based on genetic algorithms and fuzzy clustering. *Computers and Operations Research*, 35(11):3412–3438, 2008.

[SDK+95] Mary Shaw, Robert DeLine, Daniel V. Klein, Theodore L. Ross, David M. Young, and Gregory Zelesnik. Abstractions for Software Architecture and Tools to Support Them. *IEEE Transactions on Software Engineering*, 21(4):314–335, 1995.

[See96] Thomas D. Seeley. *The Wisdom of the Hive*. Harvard University Press, 1996.

[Sel59] Oliver G. Selfridge. Pandemonium: A Paradigm for Learning. In D. V. Blake and A. M. Uttley, editors, *The Mechanisation of Thought Processes*, volume 10 of *National Physical Laboratory Symposia*, pages 511–529, London, 1959. Her Majesty's Stationery Office.

[Ser04] Giovanna Di Marzo Serugendo. Trust as an Interaction Mechanism for Self-Organising Systems. In *Proceedings of the 5th International Conference on Complex Systems (ICCS 2004)*, 2004.

[Ser06] Giovanna Di Marzo Serugendo. Autonomous Systems with Emergent Behaviour. In *Handbook of Research on Nature Inspired Computing for Economy and Management*, pages 429–443. Idea Group, Inc., Hershey-PA, USA, 2006.

[SeS08] SeSAm. Shell for Simulated Agent Systems. http://www.simsesam.de, 2008.

[SFRG08] Giovanna Di Marzo Serugendo, John Fitzgerald, Alexander Romanovsky, and Nicolas Guelfi. MetaSelf - A Framework for Designing and Controlling Self-Adaptive and Self-Organising Systems. Technical Report BBKCS-08-08, School of Computer Science and Information Systems, Birkbeck College, London, UK, December 2008.

[SG96] Mary Shaw and David Garlan. *Software architecture: perspectives on an emerging discipline*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[SGHO06] Roy Sterritt, Grainne Garrity, Edward Hanna, and Patricia O'Hagan. Survivable Security Systems Through Autonomicity. In *Proceedings of the 2nd International Workshop on Radical Agent Concepts, Innovative Concepts for Autonomic and Agent-Based Systems (WRAC 2005), Revised Papers*, volume 3825 of *Lecture Notes in Computer Science*, pages 379–389. Springer, 2006.

[SGK05] Giovanna Di Marzo Serugendo, Marie-Pierre Gleizes, and Anthony Karageorgos. Self-organization in multi-agent systems. *The Knowledge Engineering Review*, 20(2):165–189, 2005.

[SGK06] Giovanna Di Marzo Serugendo, Marie-Pierre Gleizes, and Anthony Karageorgos. Self-Organisation and Emergence in MAS: An Overview. *Informatica*, 30(1):45–54, 2006.

[SH05a] Kurt Schelfthout and Tom Holvoet. A Pheromone-Based Coordination Mechanism Applied in Peer-to-Peer. In *Proceedings of the 2nd International Workshop on Agents and Peer-to-Peer Computing (AP2PC 2003), Revised and Invited Papers*, volume 2872 of *Lecture Notes in Computer Science*, pages 71–76. Springer, 2005.

[SH05b] Munindar P. Singh and Michael N. Huhns. *Service-Oriented Computing: Semantics, Processes, Agents*. John Wiley & Sons, 2005.

[Sha48] Claude E. Shannon. A Mathematical Theory of Communication. *The Bell System Technical Journal*, 27:379–423, 623–656, 1948.

[Sha95] Mary Shaw. Beyond Objects: A Software Design Paradigm Based on Process Control. *ACM Software Engineering Notes*, 20(1):27–38, 1995.

[Sha01] Cosma Rohilla Shalizi. *Causal architecture, complexity and self-organization in time series and cellular automata.* PhD thesis, University of Wisconsin at Madison, 2001.

[Sho73] Harry H. Shorey. Behavioral Responses to Insect Pheromones. *Annual Review of Entomology*, 18:349–380, 1973.

[Sho76] Harry H. Shorey. *Animal communication by pheromones.* Academic Press, New York, 1976.

[SHW08] Giovanni Samaey, Tom Holvoet, and Tom De Wolf. Engineering self-organising emergent solutions: study of equation-free macroscopic analyis. In Sven Brueckner, Paul Robertson, and Umesh Bellur, editors, *Proceedings of the 2nd IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2008)*, pages 425–434. IEEE Computer Society, 2008.

[SJF96] Douglas C. Schmidt, Ralph E. Johnson, and Mohamed Fayad. Guest Editorial for the Special Issue on Patterns and Pattern Languages. *Communications of the ACM*, 39(10):1–2, October 1996.

[SKF02] Michael Schillo, Christian Kray, and Klaus Fischer. The eager bidder problem: a fundamental problem of DAI and selected solutions. In *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2002)*, pages 599–606. ACM, 2002.

[SKRZ04] Giovanna Di Marzo Serugendo, Anthony Karageorgos, Omer F. Rana, and Franco Zambonelli, editors. *Proceedings of the 1st International Workshop on Engineering Self-Organising Systems, Nature-Inspired Approaches to Software Engineering (ESOA 2003)*, volume 2977 of *Lecture Notes in Computer Science*. Springer, 2004.

[SKS+08] Nishanth Shankaran, Xenofon D. Koutsoukos, Douglas C. Schmidt, Yuan Xue, and Chenyang Lu. Hierarchical control of multiple resources in distributed real-time and embedded systems. *Real-Time Systems*, 39(1–3):237–282, 2008.

[SL02] Tuomas Sandholm and Victor Lesser. Leveled-commitment contracting: a backtracking instrument for multiagent systems. *AI Magazine*, 23(3):89–100, 2002.

[SLB09] Yoav Shoham and Kevin Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, 2009.

[SLT08] René Schumann, Andreas D. Lattner, and Ingo J. Timm. Management-by-Exception - A Modern Approach to Managing Self-Organizing Systems. *Communications of SIWN*, 4:168–172, June 2008.

[SM02] Jim Snyder and Ronaldo Menezes. Using Logical Operators as an Extended Coordination Mechanism in Linda. In *Proceedings of the 5th International Conference on Coordination Models and Languages (COORDINATION 2002)*, volume 2315 of *Lecture Notes In Computer Science*, pages 317–331. Springer, 2002.

[SMFJZ07] Giovanna Di Marzo Serugendo, Jean-Philippe Martin-Flatin, Márk Jelasity, and Franco Zambonelli, editors. *Proceedings of the 1st IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2007)*. IEEE Computer Society, 2007.

[Smi80] Reid G. Smith. The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. *IEEE Transactions on Computers*, 29(12):1104–1113, 1980.

[SMPB05] John A. Sauter, Robert Matthews, H. Van Dyke Parunak, and Sven A. Brückner. Performance of digital pheromones for swarming vehicle control. In *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005)*, pages 903–910. ACM, 2005.

[SO06] Michael Schumacher and Sascha Ossowski. The Governing Environment. In Danny Weyns, H. Van Dyke Parunak, and Fabien Michel, editors, *Proceedings of the 2nd International Workshop Environments for Multi-Agent Systems (E4MAS 2005), Selected Revised and Invited Papers*, volume 3830 of *Lecture Notes in Computer Science*, pages 88–104. Springer, 2006.

[Sol87] Marius M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265, 1987.

[Som01] Ian Sommerville. *Software Engineering*. Pearson Studium, 2001.

[SPRR95] Yu Shen, Jean-Yves Potvin, Jean-Marc Rousseau, and Serge Roy. A computer assistant for vehicle dispatching with learning capabilities. *Annals of Operations Research*, 61(1):189–211, 1995.

[SPS04] Mikkel Sigurd, David Pisinger, and Michael Sig. Scheduling Transportation of Live Animals to Avoid the Spread of Diseases. *Transportation Science*, 38(2):197–209, 2004.

[SPTU05] Roy Sterritt, Manish Parashar, Huaglory Tianfield, and Rainer Unland. A Concise Introduction to Autonomic Computing. *Advanced Engineering Informatics*, 19(3):181–187, 2005.

[SPVG03] Katia Sycara, Massimo Paolucci, Martin Van Velsen, and Joseph Giampapa. The RETSINA MAS Infrastructure. *Autonomous Agents and Multi-Agent Systems*, 7(1-2):29–48, 2003.

[SR08] Jan Sudeikat and Wolfgang Renz. On the Encapsulation and Reuse of Decentralized Coordination Mechanisms: A Layered Architecture and Design Implications. *Communications of SIWN*, 4:140–146, 2008.

[Sri00] Rayadurgam Srikant. Control of Communication Networks. In *Perspectives in Control Engineering: Technologies, Applications, New Directions*, pages 462–488. IEEE Press, 2000.

[SS88] Torsten Söderström and Petre Stoica. *System Identification*. Prentice-Hall, 1988.

[SS95] Martin W. P. Savelsbergh and Marc Sol. The General Pickup and Delivery Problem. *Transportation Science*, 29:17–29, 1995.

[SS97] Margo Seltzer and Christopher Small. Self-Monitoring and Self-Adapting Operating Systems. In *Proceedings of the 6th Workshop on Hot Topics in Operating Systems (HOTOS 1997)*, page 124. IEEE Computer Society, 1997.

[SS98] Martin Savelsbergh and Marc Sol. Drive: Dynamic Routing of Independent Vehicles. *Operations Research*, 46(4):474–490, 1998.

[SS03] Cosma Rohilla Shalizi and Kristina Lisa Shalizi. Quantifying Self-Organization in Cyclic Cellular Automata. In Lutz Schimansky-Geier, Derek Abbott, Alexander Neiman, and Christian Van den Broeck, editors, *Noise in Complex Systems and Stochastic Dynamics (Proceedings of the SPIE)*, volume 5114, pages 108–117. SPIE, 2003.

[Sta08] StarLogo. http://education.mit.edu/starlogo, 2008.

[Ste05] Roy Sterritt. Autonomic Computing. *Innovations in Systems and Software Engineering*, 1(1):79–88, 2005.

[Ste06] Achim Stephan. The dual role of 'emergence' in the philosophy of mind and in cognitive science. *Synthese*, 151(3):485–498, Aug 2006.

[Sun02a] Sun Microsystems. Java Message Service Specification, Version 1.1. http://java.sun.com/products/jms, 2002.

[Sun02b] Sun Microsystems. N1 Software Management. http://www.sun.com/software/products/service_provisioning/index.xml, 2002.

[Swa08] Swarm. http://www.swarm.org, 2008.

[Syc98] Katia P. Sycara. Multiagent Systems. *AI Magazine*, 19(2):79–92, 1998.

[TB99] Guy Theraulaz and Eric Bonabeau. A Brief History of Stigmergy. *Artificial Life*, 5(2):97–116, 1999.

[TCW+04] Gerald Tesauro, David M. Chess, William E. Walsh, Rajarshi Das, Alla Segal, Ian Whalley, Jeffrey O. Kephart, and Steve R. White. A Multi-Agent Systems Approach to Autonomic Computing. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, pages 464–471. IEEE Computer Society, 2004.

[Ten00] David Tennenhouse. Proactive Computing. *Communications of the ACM*, 43(5):43–50, 2000.

[THRR06] Walter F. Truszkowski, Michael G. Hinchey, James L. Rash, and Christopher A. Rouff. Autonomous and Autonomic Systems: A Paradigm for Future Space Exploration Missions. *IEEE Transactions on Systems, Man, and Cybernetics - Part C: Applications and Reviews*, 36(3):279–291, May 2006.

[Tia03] Huaglory Tianfield. Multi-Agent Autonomic Architecture and Its Application in E- Medicine. In *Proceedings of the 2003 IEEE/WIC International Conference on Intelligent Agent Technology (IAT 2003)*, pages 601–604. IEEE Computer Society, 2003.

[Tia06] Huaglory Tianfield, editor. *International Transactions on Systems Science and Applications*, volume 2. SIWN Press, 2006.

[Tia07] Huaglory Tianfield, editor. *Communications of SIWN*, volume 2. SIWN Press, 2007.

[Tia08] Huaglory Tianfield, editor. *Communications of SIWN*, volume 4. SIWN Press, 2008.

[Tia09] Huaglory Tianfield, editor. *Communications of SIWN*, volume 7. SIWN Press, 2009.

[TIB09] TIBCO. TIBCO Rendezvous. http://www.tibco.com/software/messaging/rendezvous/default.jsp, 2009.

[TKA+02] Anand R. Tripathi, Neeran M. Karnik, Tanvir Ahmed, Ram D. Singh, Arvind Prakash, Vineet Kakani, Manish K. Vora, and Mukta Pathak. Design of the Ajanta System for Mobile Agent Programming. *Journal of Systems and Software*, 62(2):123–140, 2002.

[TM04] Robert Tolksdorf and Ronaldo Menezes. Using Swarm Intelligence in Linda Systems. In *Proceedings of the 4th International Workshop on Engineering Societies in the Agents World (ESAW 2003), Revised Selected and Invited Papers*, volume 3071 of *Lecture Notes in Computer Science*, pages 49–65. Springer, 2004.

[Tol96] Robert Tolksdorf. Coordinating Services in Open Distributed Systems with LAURA. In *Proceedings of the 1st International Conference on Coordination Languages and Models (COORDINATION 1996)*, volume 1061 of *Lecture Notes in Computer Science*, pages 386–402. Springer, 1996.

[TP93] Paul M. Thompson and Harilaos N. Psaraftis. Cyclic transfer algorithms for multivehicle routing and scheduling problems. *Operations Research*, 41(5):935–946, 1993.

[Tru06] Wolfgang Trumler. *Organic Ubiquitous Middleware*. Phd thesis, University of Augsburg, 2006.

[TSTN06] Graham Thomson, Graeme Stevenson, Sotirios Terzis, and Paddy Nixon. A self-managing infrastructure for ad-hoc situation determination. In *Proceedings of the 4th International Conference on Smart Homes and Health Telematics (ICOST 2006)*, pages 157–164. IOS Press, 2006.

[TV02] Paolo Toth and Daniele Vigo, editors. *The Vehicle Routing Problem*. SIAM Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2002.

[TvS06] Andrew S. Tanenbaum and Maarten van Steen. *Distributed Systems: Principles and Paradigms*. Prentice Hall, 2nd edition, 2006.

[UPn09] UPnP™ Forum. UPnP. http://www.upnp.org/, 2009.

[UPS10] UPS. UPS Fact Sheet. http://www.pressroom.ups.com/Fact+Sheets/UPS+Fact+Sheet, March 2010.

[Var79] Francisco J. Varela. *Principles of Biological Autonomy*. Elsevier Science Publishing, New York, USA, 1979.

[vB69] Ludwig von Bertalanffy. *General System Theory – Foundations, Development, Applications.* George Braziller, New York, 1969.

[VCO09] Mirko Viroli, Matteo Casadei, and Andrea Omicini. A framework for modelling and implementing self-organising coordination. In *Proceedings of the 2009 ACM Symposium on Applied Computing (SAC 2009)*, pages 1353–1360. ACM, 2009.

[VDE03] VDE/ITG/GI. Organic Computing. Position Paper (in german), Verband der Elektrotechnik / Gesellschaft für Informatik, 2003.

[vF60] Heinz von Foerster. On Self-Organizing Systems and Their Environments. In Marshall Clinton Yovits and Scott Cameron, editors, *Interdisciplinary Conference on Self-Organizing Systems*, volume 2 of *International Transactions in Computer Science and Technology and Their Application*, pages 30–50, Oxford, 1960. Pergamon Press.

[vF67] Karl von Frisch. *The Dance Language and Orientation of Bees.* Harvard University Press, 1967.

[vFJ62] Heinz von Foerster and George W. Zopf Jr., editors. *Principles of Self-Organization: Transactions of the University of Illinois Symposium on Self-Organization*, Information Systems Branch, U.S. Office of Naval Research, New York, 1962. Pergamon Press.

[VHG+07] Paul Valckenaers, Karuna Hadeli, Bart Saint Germain, Paul Verstraete, and Hendrik Van Brussel. MAS coordination and control based on stigmergy. *Computers in Industry*, 58(7):621–629, 2007.

[VHR+07] Mirko Viroli, Tom Holvoet, Alessandro Ricci, Kurt Schelfthout, and Franco Zambonelli. Infrastructures for the Environment of Multiagent Systems. *Autonomous Agents and Multi-Agent Systems*, 14(1):49–60, 2007.

[Vis06] Iris F. A. Vis. Survey of research in the design and control of automated guided vehicle systems. *European Journal of Operational Research*, 170:677–709, 2006.

[VKvB+01] Paul Valckenaers, Martin Kollingbaum, Hendrik van Brussel, Olaf Bochmann, and C. Zamfirescu. The Design of Multi-Agent Coordination and Control Systems using Stigmergy. In P. Butala and K. Ueda, editors, *Proceedings of the 3rd International Workshop on Emergent Synthesis*, 2001.

[VO06] Mirko Viroli and Andrea Omicini. Coordination as a Service. *Fundamenta Informaticae*, 73(4):507–534, 2006.

[Wal90] Arthur Wallace. *The Green Machine: Ecology and the Balance of Nature*. Basil Blackwell, Oxford, 1990.

[Wal99] Jim Waldo. The Jini architecture for network-centric computing. *Communications of the ACM*, 42(7):76–82, 1999.

[WB63] Edward O. Wilson and William H. Bossert. Chemical communication among animals. *Recent progress in hormone research*, 19:673–716, 1963.

[WBH06] Danny Weyns, Nelis Boucké, and Tom Holvoet. Gradient field-based task assignment in an AGV transportation system. In *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006)*, pages 842–849. ACM, 2006.

[WBH08] Danny Weyns, Nelis Boucké, and Tom Holvoet. A field-based versus a protocol-based approach for adaptive task assignment. *Autonomous Agents and Multi-Agent Systems*, 17(2):288–319, 2008.

[Weg96] Peter Wegner. Coordination as Constrained Interaction. In *Proceedings of the 1st International Conference on Coordination Languages and Models (COORDINATION 1996)*, volume 1061 of *Lecture Notes in Computer Science*, pages 28–33. Springer, 1996.

[Weg97] Peter Wegner. Why interaction is more powerful than algorithms. *Communications of the ACM*, 40(5):80–91, May 1997.

[Wei91] Mark Weiser. The Computer for the 21st Century. *Scientific American*, 265(3):94–104, September 1991.

[Wei99] Gerhard Weiss, editor. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, 1999.

[Wey06] Danny Weyns. *An Architecture-Centric Approach for Software Engineering with Situated Multiagent Systems*. Phd thesis, Katholieke Universiteit Leuven, 2006.

[WF71] Robert H. Whittaker and Paul P. Feeny. Allelochemics: Chemical Interactions between Species. *Science*, 171:757–770, Feb 1971.

[WFZ04] Horst Wedde, Muddassar Farooq, and Yue Zhang. BeeHive: An Efficient Fault-Tolerant Routing Algorithm Inspired by Honey Bee Behavior. In Marco Dorigo, Mauro Birattari, Christian Blum, Luca Maria Gambardella, Francesco Mondada, and Thomas Stützle, editors, *Proceedings of the 4th International WorkshopAnt Colony Optimization and Swarm Intelligence (ANTS 2004)*, volume 3172 of *Lecture Notes in Computer Science*, pages 83–94. Springer, 2004.

[WH04] Danny Weyns and Tom Holvoet. A Formal Model for Situated Multi-Agent Systems. *Fundamenta Informaticae*, 63(2–3):125–158, 2004.

[Whe26] William Morton Wheeler. Emergent Evolution and the Social. *Science*, 64(1662):433–440, November 1926.

[WHE+08] Danny Weyns, Robrecht Haesevoets, Bart Van Eylen, Alexander Helleboogh, Tom Holvoet, and Wouter Joosen. Endogenous versus exogenous self-management. In *Proceedings of the 2008 International Workshop on Software Engineering for Adaptive and Self-managing Systems (SEAMS 2008)*, pages 41–48. ACM, 2008.

[WHHS09] Danny Weyns, Alexander Helleboogh, Tom Holvoet, and Michael Schumacher. The Agent Environment in Multi-Agent Systems: A Middleware Perspective. *Multiagent and Grid Systems*, 5(1):92–108, 2009.

[Whi70a] Robert H. Whittaker. *Communities and Ecosystems*. Macmillan Company, New York, 1970.

[Whi70b] Robert H. Whittaker. The biochemical ecology of higher plants. In Ernest Sondheimer and John B. Simeone, editors, *Chemical Ecology*, pages 43–70. Academic Press, New York, 1970.

[Whi79] Alfred North Whitehead. *Process and Reality: An Essay in Cosmology (Gifford Lectures Delivered in the University of Edinburgh During the Session 1927-28)*. Free Press, corrected edition, 1979.

[Whi07] Whitestein Technologies. Agent-oriented Development Methodology. http://www.whitestein.com/adem, 2007.

[Whi09] Whitestein Technologies. Living Systems. http://www.whitestein.com/autonomic-technology-platform/ls-ts-living-systems-technology-suite, March 2009.

[WHW+04] Steve R. White, James E. Hanson, Ian Whalley, David M. Chess, and Jeffrey O. Kephart. An Architectural Approach to Autonomic Computing. In *Proceedings of the 1st International Conference on Autonomic Computing (ICAC 2004)*, pages 2–9. IEEE Computer Society, 2004.

[Wie65] Norbert Wiener. *Cybernetics: or the Control and Communication in the Animal and the Machine*. MIT Press, 2nd edition, 1965.

[Wil71] Edward O. Wilson. *The Insect Societies*. Belknap Press, Cambridge, MA, 1971.

[Wil75] Edward O. Wilson. *Sociobiology: The New Synthesis*. Harvard University Press, 1975.

[WJ95] Michael J. Wooldridge and Nick R. Jennings. Intelligent Agents: Theory and Practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.

[WMA10] Danny Weyns, Sam Malek, and Jesper Andersson. FORMS: a FOrmal Reference Model for Self-adaptation. In *Proceedings of the 7th International Conference on Autonomic Computing and Communications (ICAC 2010)*, pages 205–214. ACM, 2010.

[WMAS10] Danny Weyns, Sam Malek, Jesper Andersson, and Bradley Schmerl, editors. *Proceedings of the 2nd International Workshop on Self-Organizing Architectures (SOAR 2010)*. ACM, 2010.

[WMdLA10] Danny Weyns, Sam Malek, Rogerio de Lemos, and Jesper Andersson, editors. *Proceedings of the 1st International Workshop on Self-Organizing Architectures (SOAR 2009), Revised Selected and Invited Papers*, volume 6090 of *Lecture Notes in Computer Science*. Springer, 2010.

[Wol96] Elmar Wolfstetter. Auctions: An Introduction. *Journal of Economic Surveys*, 10(4):367–420, December 1996.

[WOO07] Danny Weyns, Andrea Omicini, and James Odell. Environment as a first class abstraction in multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 14(1):5–30, 2007.

[Woo09] Michael Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley & Sons, 2nd edition, 2009.

[WPM+05] Danny Weyns, H. Van Dyke Parunak, Fabien Michel, Tom Holvoet, and Jacques Ferber. Environments for Multiagent Systems State-of-the-Art and Research Challenges. In *Proceedings of the 1st International Workshop on Environments for Multi-Agent Systems (E4MAS 2004), Revised Selected Papers*, volume 3374 of *Lecture Notes in Computer Science*, pages 1–47. Springer, 2005.

[WPT03] Roy Want, Trevor Pering, and David Tennenhouse. Comparing Autonomic and Proactive Computing. *IBM Systems Journal*, 42(1):129–135, 2003.

[WSDG01] W. Andy Wright, Robert E. Smith, Martin Danek, and Pillip Greenway. A Generalisable Measure of Self-Organisation and Emergence. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN 2001)*, volume 2130 of *Lecture Notes In Computer Science*, pages 857–864. Springer, 2001.

[WSHL05] Danny Weyns, Kurt Schelfthout, Tom Holvoet, and Tom Lefever. Decentralized control of E'GV transportation systems. In *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005)*, pages 67–74. ACM, 2005.

[WVH06] Danny Weyns, Giuseppe Vizzari, and Tom Holvoet. Environments for Situated Multi-agent Systems: Beyond Infrastructure. In Danny Weyns, H. Van Dyke Parunak, and Fabien Michel, editors, *Proceedings of the 2nd International Workshop on Environments for Multi-Agent Systems (E4MAS 2005), Selected Revised and Invited Papers*, volume 3830 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2006.

[WWWMM01] Michael P. Wellman, William E. Walsha, Peter R. Wurman, and Jeffrey K. MacKie-Mason. Auction Protocols for Decentralized Scheduling. *Games and Economic Behavior*, 35(1-2):271–303, 2001.

[Wya03] Tristram D. Wyatt. *Pheromones and Animal Behaviour: Communication by Smell and Taste*. Cambridge University Press, 2003.

[XCRA03] Hang Xu, Zhi-Long Chen, Srinivas Rajagopal, and Sundar Arunapuram. Solving a Practical Pickup and Delivery Problem. *Transportation Science*, 37(3):347–364, 2003.

[XRa08] XRaptor. http://www.informatik.uni-mainz.de/~polani/XRaptor/XRaptor.html, 2008.

[XSSL06] Yang Xu, Paul Scerri, Katia Sycara, and Michael Lewis. Comparing market and token-based coordination. In *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006)*, pages 1113–1115. ACM, 2006.

[XSY+05] Yang Xu, Paul Scerri, Bin Yu, Steven Okamoto, Michael Lewis, and Katia P. Sycara. An integrated token-based algorithm for scalable coordination. In Frank Dignum, Virginia Dignum, Sven Koenig, Sarit Kraus, Munindar P. Singh, and Michael Wooldridge, editors, *Proceedings of the 4rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005)*, pages 407–414. ACM, 2005.

[XW05] Rui Xu and Donald Wunsch II. Survey of Clustering Algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, 2005.

[Yan05] Xin-She Yang. Engineering Optimizations via Nature-Inspired Virtual Bee Algorithms. In *Proceedings of the 1st International Work-Conference on the Interplay Between Natural and Artificial Compu-*

tation (IWINAC 2005), volume 3562 of *Lecture Notes in Computer Science*, pages 317–323. Springer, 2005.

[Yan08] Xin-She Yang. *Nature-Inspired Metaheuristic Algorithms*. Luniver Press, 2008.

[YC60] Marshall Clinton Yovits and Scott Cameron, editors. *Interdisciplinary Conference on Self-Organizing Systems*, volume 2 of *International Transactions in Computer Science and Technology and Their Application*, Oxford, 1960. Pergamon Press.

[YK96] Makoto Yokoo and Yasuhiko Kitamura. Multiagent real-time A* with selection: Introducing competition in cooperative search. In Mario Tokoro, editor, *Proceedings of 2nd International Conference on Multi-Agent Systems (ICMAS 1996)*, pages 409–416. AAAI Press, 1996.

[Zan36] Enoch Zander. *Bienenkunde im Obstbau*. Eugen Ulmer Verlag, Stuttgart, 1936.

[ZJW03] Franco Zambonelli, Nicholas R. Jennings, and Michael Wooldridge. Developing multiagent systems: The Gaia methodology. *ACM Transactions on Software Engineering and Methodology*, 12(3):317–370, 2003.

[ZLL09] Dan Zhenggang, Cai Linning, and Zheng Li. Improved Multi-Agent System for the Vehicle Routing Problem with Time Windows. *Tsinghua Science & Technology*, 14(3):407–412, 2009.

[ZO04] Franco Zambonelli and Andrea Omicini. Challenges and Research Directions in Agent-Oriented Software Engineering. *Autonomous Agents and Multi-Agent Systems*, 9(3):253–283, 2004.

[ZP03] Franco Zambonelli and H. Van Dyke Parunak. Towards a paradigm change in computer science and software engineering: a synthesis. *The Knowledge Engineering Review*, 18(4):329–342, December 2003.

[ZW07] Michael Zapf and Thomas Weise. Offline Emergence Engineering For Agent Societies. In *Proceedings of the 5th European Workshop on Multi-Agent Systems (EUMAS 2007)*, 2007.

[ZYK07] Teruyoshi Zenmyo, Hideki Yoshida, and Tetsuro Kimura. A self-healing technique based on encapsulated operation knowledge. *Cluster Computing*, 10(4):385–394, 2007.

List of Abbreviations

AC	Autonomic Computing
ACL	Agent-Communication Language
ADL	Architectural Description Language
AGV	Automated Guided Vehicle
AM	Autonomic Manager
ANS	Autonomous Nervous System
AOSE	Agent-Oriented Software Engineering
AUML	Agent Unified Modeling Language
CAS	Complex Adaptive System
CNP	Contract Net Protocol
DAC	Decentralized Autonomic Computing
DAI	Distributed Artificial Intelligence
DARP	Dial-A-Ride Problem
DIC	Digital Infochemical Coordination
dod	Degree of Dynamism
DSI	Dynamic Systems Initiative
ECA	Event-condition-action
edod	Effective Degree of Dynamism
EIA	Efficiency Improvement Advisor
GPDP	General Pickup and Delivery Problem
IBC	Infochemical-based coordination
MAS	Multi-Agent System
MBE	Management-by-exception
MPC	Model Predictive Control
NGDC	Next-Generation Data Center
NMPC	Nonlinear Model Predictive Control
O/C	Observer/Controller
OPEX	Operational Expenditure
OR	Operations Research
PBC	Pheromone-based Coordination
PDP	Pickup and Delivery Problem
PIC	Pollination-Inspired Coordination
QoS	Quality of Service
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SE	Software Engineering
SENSES	Simulator for EfficieNt Self-Organizing Emergent Systems

SLA	Service Level Agreement
SLC	Sequential Leader Clustering
SOA	Service-Oriented Architecture
SOS	Self-organizing System
SuOC	System under observation/control
TCO	Total Cost of Ownership
TOD	Transportation on Demand
TSP	Traveling Salesman Problem
UML	Unified Modeling Language
UP	Unified Process
WSN	Wireless Sensor Network

List of Symbols

A	Set of Agents
Ac	Action chain of an agent
Act	Set of actions an agent can perform
Ag	Agent
Ag_{EIA}	Advisor agent
Ag^{inf}	Infochemical-processing agent
Ag^{r}	Rule-applying agent
C	Set of connections
Cl	Cluster of tasks
D	Diffusion coefficient of a natural chemical
Dat	Set of possible values in internal data area of an agent
DS	Set of delivery stations
Ed	Euclidean distance
Env	Environment
Env^{inf}	Infochemical environment
Fl	Set of flowers associated with a plant
$GHist$	Set of possible global agent histories
$Hist$	Set of possible local agent histories
K	Behavioral threshold concentration of natural organisms
L	Set of locations
LC	List of clusters
M	Map
P	Problem
PS	Set of pickup stations
Q	Load of vehicle when leaving a station
R	Set of transportation requests
$Rule$	Set of rules
S	Set of stations
Sit	Set of situations an agent can be in
Sol	Set of solutions
T	Set of tasks
T^{rec}	Set of recurring tasks
U	Concentration of a volatile chemical in nature
V	Set of vehicles

W	Emission rate of vapor from a natural emitter
α	Allomone
χ	Infochemical a pollinator agent is following
δ	Diffusion coefficient of an infochemical
ϵ	Emitter of an infochemical
η	Weight factor
γ^{thresh}	Threshold concentration of an infochemical
γ	Current concentration of an infochemical
ι	Infochemical
κ	Kairomone
λ	Weight factor
μ	Weight factor
ν	Weight factor
ω	Weight factor
ϖ	Weight factor
φ	Pheromone
ψ	Individual information encapsulated in an infochemical
ς	Synomone
τ	Weight factor
θ	Type of an agent
υ	Weight factor
$\mathcal{A}$	Set of allomones
$\mathcal{I}$	Set of infochemicals
$\mathcal{K}$	Set of kairomones
$\mathcal{P}$	Set of pheromones
$\mathcal{S}$	Set of synomones
act	Action an agent can perform
as	Assignment function
c	Connection
cap	Capacity
$cent$	Centroid function
cl	Medoid of a cluster
$clustthresh$	Threshold parameter
$comp$	Compartment
cr	Conflict resolution mechanism
dat	Value in internal data area of an agent
dpg	Pollen grains desired by a flower
dt	Deadlock time
dt	Deadlock time threshold

ef	Evaporation factor
emr	Emission rate of an agent Ag^{inf}
er	Evaporation rate
$erfc$	Complementary error function
f_{Ag}	Decision function of an agent
fl	Flower
ft	Follow time, i. e. the time an agent follows the same synomone
$ghist$	global agent history
$hist$	Local agent history
it	Idle time
l	Location
$minocc$	Threshold parameter
opg	Pollen grains offered by a pollinator to a flower
pf	Propagation factor
pg	Pollen grains
ppg	Pollen grains provided by a flower
pr	Propagation rate
q	Loadsize
$qual$	Quality function
$qualthresh$	Threshold parameter
r	Exception rule
rc	Concentration of the reward provided by a flower
rpg	Pollen grains requested by a pollinator from a flower
run	Run (sequence of tasks)
s	Station
sim	Similarity function
sit	Situation an agent can be in
sol	Solution
st	Service time
t	Iteration (point in time)
ta	Task
u	Utility of following an infochemical
ut	Unsuccessful inspection time
v	Vehicle

List of Figures

List of Tables

Appendix A

XML Schemas

Listing A.1: XML Schema for main configuration files for SENSES

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://pvs.informatik.uni-augsburg.de/senses-data"
  xmlns:senses="http://pvs.informatik.uni-augsburg.de/senses-data"
  elementFormDefault="qualified">

  <element name="configuration" type="senses:configuration" />

  <complexType name="configuration">
    <sequence>
      <element name="default" type="senses:default" maxOccurs="1"
        minOccurs="0" />
      <element name="experiment" type="senses:experiment"
        maxOccurs="1" minOccurs="0" />
      <element name="logging" type="senses:logging" maxOccurs="1"
        minOccurs="0" />
    </sequence>
  </complexType>

  <complexType name="experiment">
    <sequence>
      <element name="global" type="senses:global" maxOccurs="1"
        minOccurs="0" />
      <element name="run" type="senses:run" minOccurs="1"
        maxOccurs="unbounded" />
      <element name="eca" type="senses:eca" maxOccurs="1"
        minOccurs="0" />
    </sequence>
    <attribute name="name" type="string" />
    <attribute name="type" type="string" />
    <attribute name="optimize" type="boolean" />
    <attribute name="record" type="boolean" />
    <attribute name="protocol" type="boolean" />
    <attribute name="number-of-runs" type="int"></attribute>
    <attribute name="number-of-runs-learning" type="int" />
  </complexType>

  <complexType name="default">
    <sequence>
      <element name="settings" type="senses:settings" maxOccurs="1"
```

```
      minOccurs="0" />
    <element name="parameters" type="senses:parameters"
      maxOccurs="1" minOccurs="0" />
    <element name="optimization" type="senses:optimization"
      maxOccurs="1" minOccurs="0" />
    <element name="general" type="senses:general" maxOccurs="1"
      minOccurs="0" />
  </sequence>
</complexType>

<complexType name="global">
  <sequence>
    <element name="setup" type="senses:setup" maxOccurs="1"
      minOccurs="0" />
    <element name="settings" type="senses:settings" maxOccurs="1"
      minOccurs="0" />
    <element name="parameters" type="senses:parameters"
      maxOccurs="1" minOccurs="0" />
  </sequence>
</complexType>

<complexType name="run">
  <sequence>
    <element name="setup" type="senses:setup" maxOccurs="1"
      minOccurs="0" />
    <element name="settings" type="senses:settings" maxOccurs="1"
      minOccurs="0" />
    <element name="parameters" type="senses:parameters"
      maxOccurs="1" minOccurs="0" />
  </sequence>
</complexType>

<complexType name="setup">
  <sequence>
    <element name="problem-domain" type="string" maxOccurs="1"
      minOccurs="1" />
    <element name="environment" type="senses:environment"
      maxOccurs="1" minOccurs="1" />
    <element name="scenario" type="senses:scenario"
      maxOccurs="1" minOccurs="1" />
    <element name="coordination-mechanism" type="string"
      maxOccurs="1" minOccurs="1" />
  </sequence>
</complexType>

<complexType name="environment">
  <simpleContent>
    <extension base="string">
      <attribute name="path" type="string"></attribute>
    </extension>
  </simpleContent>
</complexType>

<complexType name="scenario">
```

```
    <simpleContent>
      <extension base="string">
        <attribute name="path" type="string"></attribute>
      </extension>
    </simpleContent>
  </complexType>

  <complexType name="settings">
    <sequence>
      <element name="iteration-limit" type="int" maxOccurs="1"
        minOccurs="0" />
      <element name="number-of-agents" type="int" maxOccurs="1"
        minOccurs="0" />
      <element name="task-generation-number" type="int" maxOccurs="1"
        minOccurs="0" />
      <element name="task-generation-limit" type="int" maxOccurs="1"
        minOccurs="0" />
      <element name="task-generation-probability" type="float"
        maxOccurs="1" minOccurs="0" />
    </sequence>
  </complexType>

  <complexType name="parameters">
    <all>
      <element name="PIC" type="senses:picParameters" />
    </all>
  </complexType>

  <complexType name="optimization">
    <sequence>
      <element name="optimize" type="boolean" maxOccurs="1"
        minOccurs="0" />
      <element name="number-of-runs" type="int" maxOccurs="1"
        minOccurs="0" />
      <element name="number-of-runs-learning" type="int" maxOccurs="1"
        minOccurs="0" />
    </sequence>
  </complexType>

  <complexType name="general">
    <sequence>
      <element name="record" type="boolean" maxOccurs="1"
        minOccurs="0" />
      <element name="protocol" type="boolean" maxOccurs="1"
        minOccurs="0" />
      <element name="logging" type="boolean" maxOccurs="1"
        minOccurs="0" />
    </sequence>
  </complexType>

  <complexType name="picParameters">
    <sequence>
      <element name="emission-concentration" type="float" />
      <element name="reward-concentration" type="float" />
```

```
      <element name="reward-concentration-variation" type="float" />
      <element name="idle-time" type="float" />
      <element name="emission-rates" type="senses:emissionRates" />
      <element name="pheromones" type="senses:infochemicalParameters"/>
      <element name="allomones" type="senses:infochemicalParameters"/>
      <element name="kairomones" type="senses:infochemicalParameters"/>
      <element name="synomones" type="senses:infochemicalParameters"/>
    </sequence>
  </complexType>

  <complexType name="emissionRates">
    <sequence>
      <element name="pollinator-agents" type="float" />
      <element name="pollenizer-agents" type="float" />
      <element name="hive-agents" type="float" />
    </sequence>
  </complexType>

  <complexType name="infochemicalParameters">
    <sequence>
      <element name="evaporation-factor" type="float" />
      <element name="diffusion-coefficient" type="float" />
      <element name="threshold-concentration" type="float" />
    </sequence>
  </complexType>

  <complexType name="eca">
    <sequence>
      <element name="rules" type="senses:rules" />
      <element name="output" type="senses:output" />
      <element name="precalculated-distances"
        type="senses:precalculated-distances" />
      <element name="ga" type="senses:ga" />
    </sequence>
  </complexType>

  <complexType name="rules">
    <attribute name="load-rules" type="string" />
    <attribute name="on-run" type="int" />
    <attribute name="filename" type="string" />
  </complexType>

  <complexType name="output">
    <attribute name="enabled" type="boolean" />
    <attribute name="filename" type="string" />
  </complexType>

  <complexType name="precalculated-distances">
    <attribute name="load-values" type="boolean" />
    <attribute name="save-values" type="boolean" />
    <attribute name="filename" type="string" />
  </complexType>

  <complexType name="ga">
```

```
    <sequence>
      <element name="heap" type="senses:heap" />
      <element name="benchmark" type="senses:benchmark" />
      <element name="population" type="senses:population" />
      <element name="social-collapse" type="senses:social-collapse" />
    </sequence>
    <attribute name="run-length" type="string" />
    <attribute name="use-cache" type="boolean" />
  </complexType>

  <complexType name="heap">
    <attribute name="min-heap" type="int" />
    <attribute name="max-heap" type="int" />
  </complexType>

  <complexType name="benchmark">
    <attribute name="enabled" type="boolean" />
    <attribute name="eval" type="double" />
    <attribute name="times" type="int" />
  </complexType>

  <complexType name="population">
    <attribute name="new-random" type="int" />
    <attribute name="mut-over-cross" type="int" />
  </complexType>

  <complexType name="social-collapse">
    <attribute name="enabled" type="boolean" />
    <attribute name="after" type="int" />
    <attribute name="perc-keep" type="int" />
  </complexType>

  <complexType name="logging">
    <sequence>
      <element name="logger" type="senses:logger"
        maxOccurs="unbounded" minOccurs="0" />
    </sequence>
  </complexType>

  <complexType name="logger">
    <attribute name="name" type="string" />
    <attribute name="logging" type="boolean" />
    <attribute name="log-to-file" type="boolean" />
  </complexType>
</schema>
```

Listing A.2: XML Schema for environment configuration files for SENSES

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://pvs.informatik.uni-augsburg.de/senses-data"
  xmlns:senses="http://pvs.informatik.uni-augsburg.de/senses-data"
  elementFormDefault="qualified">
```

```
  <element name="environment" type="senses:environment" />

  <complexType name="environment">
    <sequence maxOccurs="1" minOccurs="0">
      <element name="locations" type="senses:list-of-locations" />
      <element name="connections" type="senses:list-of-connections" />
    </sequence>
    <attribute name="name" type="string" />
    <attribute name="width" type="int" />
    <attribute name="height" type="int" />
    <attribute name="scale" type="int" />
    <attribute name="type" type="string" />
  </complexType>

  <complexType name="list-of-locations">
    <sequence>
      <element maxOccurs="unbounded" minOccurs="0"
        name="location" type="senses:location" />
    </sequence>
  </complexType>

  <complexType name="list-of-connections">
    <sequence>
      <element maxOccurs="unbounded" minOccurs="0"
        name="connection" type="senses:connection" />
    </sequence>
  </complexType>

  <complexType name="location">
    <attribute name="id" type="int" />
    <attribute name="x-pos" type="int" />
    <attribute name="y-pos" type="int" />
  </complexType>

  <complexType name="connection">
    <attribute name="id" type="int" />
    <attribute name="from" type="int" />
    <attribute name="to" type="int" />
  </complexType>

</schema>
```

Listing A.3: XML Schema for PDP scenario configuration files for SENSES

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://pvs.informatik.uni-augsburg.de/senses-data"
  xmlns:senses="http://pvs.informatik.uni-augsburg.de/senses-data"
  elementFormDefault="qualified">

  <element name="scenario" type="senses:pdpScenario" />

  <complexType name="pdpScenario">
    <sequence>
```

```
      <element name="request-types" type="senses:listOfRequestTypes" />
      <element name="vehicle-types" type="senses:listOfVehicleTypes" />
      <element name="depots" type="senses:listOfDepots" />
      <element name="stations" type="senses:listOfStations" />
      <element name="vehicles" type="senses:listOfVehicles" />
      <element name="requests" type="senses:listOfRequests"
        minOccurs="0" maxOccurs="1" />
    </sequence>
  </complexType>

  <complexType name="listOfVehicleTypes">
    <sequence>
      <element name="vehicle-type" type="senses:vehicleType"
        minOccurs="1" maxOccurs="unbounded" />
    </sequence>
  </complexType>

  <complexType name="listOfRequestTypes">
    <sequence>
      <element name="request-type" type="senses:requestType"
        minOccurs="1" maxOccurs="unbounded" />
    </sequence>
  </complexType>

  <complexType name="listOfDepots">
    <sequence>
      <element name="depot" type="senses:depot"
        minOccurs="0" maxOccurs="unbounded" />
    </sequence>
  </complexType>

  <complexType name="listOfStations">
    <sequence>
      <element name="station" type="senses:station"
        minOccurs="0" maxOccurs="unbounded" />
    </sequence>
  </complexType>

  <complexType name="listOfVehicles">
    <sequence>
      <element name="vehicle" type="senses:vehicle"
        minOccurs="0" maxOccurs="unbounded" />
    </sequence>
  </complexType>

  <complexType name="listOfRequests">
    <sequence>
      <element name="request" type="senses:request"
        minOccurs="0" maxOccurs="unbounded" />
    </sequence>
  </complexType>

  <complexType name="vehicleType">
    <sequence>
```

```
      <element name="sub-types" type="senses:listOfVehicleTypes"
        minOccurs="0" maxOccurs="1" />
      <element name="super-types" type="senses:listOfVehicleTypes"
        minOccurs="0" maxOccurs="1" />
      <element name="request-types" type="senses:listOfRequestTypes"
        minOccurs="0" maxOccurs="1" />
    </sequence>
    <attribute name="name" type="string" />
  </complexType>

  <complexType name="requestType">
    <sequence>
      <element name="sub-types" type="senses:listOfRequestTypes"
        minOccurs="0" maxOccurs="1" />
      <element name="super-types" type="senses:listOfRequestTypes"
        minOccurs="0" maxOccurs="1" />
    </sequence>
    <attribute name="name" type="string" />
  </complexType>

  <complexType name="depot">
    <sequence>
      <element name="vehicle-types" type="senses:listOfVehicleTypes" />
    </sequence>
    <attribute name="id" type="int" />
    <attribute name="position" type="int" use="optional" />
    <attribute name="x-pos" type="int" use="optional" />
    <attribute name="y-pos" use="optional" />
  </complexType>

  <complexType name="station">
    <sequence>
      <element name="request-types" type="senses:listOfRequestTypes"
      maxOccurs="1" minOccurs="0"/>
    </sequence>
    <attribute name="id" type="int" />
    <attribute name="position" type="int" use="optional" />
    <attribute name="x-pos" type="int" use="optional" />
    <attribute name="y-pos" use="optional" />
    <attribute name="type" use="optional">
      <simpleType>
        <restriction base="string">
          <enumeration value="pickup" />
          <enumeration value="delivery" />
          <enumeration value="universal" />
        </restriction>
      </simpleType>
    </attribute>
  </complexType>

  <complexType name="vehicle">
    <attribute name="id" type="int" />
    <attribute name="position" type="int" use="optional" />
    <attribute name="x-pos" type="int" use="optional" />
```

```
    <attribute name="y-pos" use="optional" />
    <attribute name="capacity" type="int" />
    <attribute name="speed" type="int" />
    <attribute name="type" type="string" />
  </complexType>

  <complexType name="request">
    <sequence>
      <element name="pickup-station" type="senses:requestStation" />
      <element name="delivery-station" type="senses:requestStation" />
    </sequence>
    <attribute name="id" type="int" />
    <attribute name="loadsize" type="int" />
    <attribute name="service-time" type="int" />
    <attribute name="probability" type="float"
      use="optional" />
    <attribute name="type" type="string" />
  </complexType>

  <complexType name="requestStation">
    <attribute name="id" type="int" />
    <attribute name="earliest" type="int" />
    <attribute name="latest" type="int" />
  </complexType>

</schema>
```

Listing A.4: XML Schema for PDP result files for SENSES

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://pvs.informatik.uni-augsburg.de/senses-data"
  xmlns:senses="http://pvs.informatik.uni-augsburg.de/senses-data"
  elementFormDefault="qualified">

  <element name="result" type="senses:pdpResult" />

  <complexType name="pdpResult">
    <sequence>
      <element name="run" type="senses:run"
        maxOccurs="unbounded" minOccurs="1" />
    </sequence>
    <attribute name="experiment" type="string"></attribute>
  </complexType>

  <complexType name="run">
      <sequence>
        <element name="iterations" type="int"/>
        <element name="tasks-upcoming" type="int"/>
        <element name="tasks-announced" type="int"/>
        <element name="tasks-in-process" type="int"/>
        <element name="tasks-finished" type="int"/>
        <element name="agents-on-the-move" type="int"/>
        <element name="throughput" type="float"/>
```

```
        <element name="total-travel-costs" type="float"/>
        <element name="total-tolerated-waiting-time" type="int"/>
        <element name="total-non-tolerated-waiting-time" type="int"/>
        <element name="average-tolerated-waiting-time" type="float"/>
        <element name="average-non-tolerated-waiting-time"
          type="float"/>
        <element name="delayed-pickups" type="int"/>
        <element name="delayed-deliveries" type="int"/>
        <element name="total-infochemicals" type="int"/>
        <element name="total-pheromones" type="int"/>
        <element name="total-allomones" type="int"/>
        <element name="total-kairomones" type="int"/>
        <element name="total-synomones" type="int"/>
        <element name="emitted-infochemicals" type="int"/>
        <element name="emitted-pheromones" type="int"/>
        <element name="emitted-allomones" type="int"/>
        <element name="emitted-kairomones" type="int"/>
        <element name="emitted-synomones" type="int"/>
      </sequence>
      <attribute name="number" type="int"/>
    </complexType>
</schema>
```

Holger Kasinger

Curriculum Vitae

Personal Data

Day of birth	December 20, 1979
Place of birth	Munich
Nationality	German
Family status	Married, two children

Education

02/2005 – 07/2010	**Doctorate**, *University of Augsburg*, Augsburg, Faculty of Applied Computer Science, Supervisor: Prof. Dr. Bernhard Bauer.
10/2000 – 01/2005	**Studies**, *University of Augsburg*, Augsburg, Diploma course: Applied Computer Science Degree: Diploma in Computer Science.
08/1999 – 08/2000	**Community service**, *Nachsorge Zentrum Augsburg*, Augsburg.
09/1990 – 07/1999	**Secondary school**, *Rudolf-Diesel-Gymnasium*, Augsburg, Degree: general qualification for university entrance.