PosoMAS: an extensible, modular SE process for open self-organising systems

Jan-Philipp Steghöfer, Hella Seebach, Benedikt Eberhardinger, Wolfgang Reif

Angaben zur Veröffentlichung / Publication details:

Steghöfer, Jan-Philipp, Hella Seebach, Benedikt Eberhardinger, and Wolfgang Reif. 2014. "PosoMAS: an extensible, modular SE process for open self-organising systems." In *PRIMA 2014: Principles and Practice of Multi-Agent Systems: 17th International Conference, Gold Coast, QLD, Australia, December 1-5, 2014, proceedings*, edited by Hoa Khanh Dam, Jeremy Pitt, Yang Xu, Guido Governatori, and Takayuki Ito, 1–17. Cham: Springer. https://doi.org/10.1007/978-3-319-13191-7_1.

PosoMAS: An Extensible, Modular SE Process for Open Self-organising Systems

Jan-Philipp Steghöfer, Hella Seebach, Benedikt Eberhardinger, and Wolfgang Reif

Institute for Software & Systems Engineering, Augsburg University, Germany
{firstname.lastname}@informatik.uni-augsburg.de

Abstract. This paper introduces PosoMAS, the Process for open, self-organising Multi-Agent Systems. The process is composed of a number of practices, reusable and customisable building parts, and integrated into the lifecycle of the Open Unified Process to yield an iterative, incremental software engineering process tailored to open self-organising systems. The individual practices are introduced and their interplay described. We evaluate PosoMAS in two case studies and provide a qualitative comparison with existing AOSE processes.

1 Requirements for Agent-Oriented Software Engineering Processes

If a system has to be open and has to exhibit self-organisation, principled software engineering techniques become even more important. For instance, in such cases, the benevolence assumption, i.e., the assumption that the individual agents contribute to reaching an overall system goal, can no longer be maintained. The dynamics of self-organisation and the potential negative emergent effects are thus coupled with self-interested, erratic, and even potentially malevolent agents that still have to be integrated in the system. Examples for domains that exhibit such effects are energy management [1] and open grid computing [2].

Our previous scientific contributions (refer to, e.g., [1, 3–5]) have dealt with these issues without being embedded in a methodology for the principled design of such systems.

PosoMAS, the **P**rocess for **o**pen **s**elf-**o**rganising **M**ulti**a**gent **S**ystems, has been designed to remedy this situation. It addresses a number of requirements outlined below that are motivated by the need to make multi-agent technology and self-organisation principles available to software engineers and by the specific characteristics of open, self-organising systems. We do not claim that the process is the be-all and end-all of agent-oriented software engineering approaches but addresses specific circumstances under which it is applicable. If a project does not adhere to the assumptions made by PosoMAS or requires additional aspects, other processes might be more suitable. However, due to its modular design, PosoMAS can be adapted to suit the needs of a specific product or development team. Based on an analysis of existing agent-oriented software engineer-

ing (AOSE) processes (cf. Section 2) and our own experience with self-organising systems, we identified the following requirements:

Extensibility and Customisability. The methodology must be extensible. It must be possible to combine it with different process models and to customise it for specific situations. This means that it must be possible to use the elements of the method in an agile context (e.g., in a specialised Scrum process) as well as in a heavy-weight context (e.g., the still pervasive waterfall method).

Independence from Architectures or Tools. The internal architecture of the agents (such as BDI) and the concrete implementation platform should play no role in the high-level design part of the process. Likewise, the modelling language should not be pre-determined to allow designers with a regular software engineering background to use tools that they know and understand.

Clear Separation of Different Architecture Domains. To accommodate open systems and separate design teams, the process has to provide aids that allow the separate definition of interfaces, data models, and interactions so that other development teams know how the agents should behave in the system, interact with other agents, and with the system as a whole.

Special Focus on Interactions between Agents and Agent Organisations. The dynamics of an open self-organising multi-agent system are defined by the interactions between agents within organisations. The behaviour of the individual agent within an organisation determines the fitness for purpose of the organisation and of its ability to reach its goal within the system. Organisational structure also affects scalability of the final system. In addition, self-organisation functionality is usually a result of bottom-up interactions that have to be consolidated with the top-down requirements [3].

We adopt the principles of standard software engineering methods such as the OpenUP (PosoMAS uses the Eclipse Process Framework (EPF) practices library that contains OpenUP building blocks), that promote, e.g., reuse, evolutionary design, shared vision. These principles are documented, e.g., in [6] for the Rational Unified Process (RUP), a commercial methodology that introduced many of the features present in modern processes. Arguably, the value of processes such as RUP stems mostly from their extensive documentation of SE practices and guidelines. These are used in a situational method engineering (SME) approach for the creation of a customised software engineering methodology from these reusable assets. Likewise, PosoMAS provides such assets containing a wealth of information on AOSE with a focus on self-organisation and adaptation which continues to grow as the process matures. These building blocks are collected in a *method library* or *method repository* [7] which is available at `http://posomas.isse.de`, along with the process description and an example.

This paper introduces PosoMAS, relates and compares it to existing AOSE methodologies in Section 2, describes the practices that make up the method content in Section 3 as well as the life cycle it uses in Section 4. Since the format of a paper is insufficient to describe a comprehensive methodology in full detail, the reader is advised to peruse the detailed process description at

http://posomas.isse.de. The website also offers the process description for use in the EPF Composer. Finally, Section 5 compares PosoMAS with existing processes, discusses benefits and lessons learned in simulated development efforts. The paper closes with a discussion of future work.

2 Characteristics of Existing AOSE Methodologies

This section gives an overview of current AOSE methodologies, pointing out their unique characteristics. Apart from the original papers on the methodologies, we also use content provided by attempts to compare methodologies. Such comparative studies (e.g., [8]) are, however, to be taken with a grain of salt, since the set of evaluation criteria used are not necessarily agreed-upon standards. Since such standards are missing, however, such studies currently provide the only reference point for comparing AOSE methodologies. The processes selected below have been mainly chosen due to the currentness of the published method content. A recent overview of agent-oriented design processes is presented in [9] where a number of processes are cast in the IEEE FIPA Process Documentation Template but the book offers no *new* method content (e.g., for Gaia or Tropos) or a qualitative comparison of the methodologies.

The **Prometheus** methodology [10] combines specification, design, and implementation in a detailed process and is commonly accepted as one of the most mature AOSE approaches (cf. [11–13]). Prometheus uses a bottom-up approach for the development of multi-agent systems with BDI agents. While the focus on BDI is often lauded [12, 13], some authors criticise that this constitutes a restriction of application domains [11]. According to [10], however, only a subset of Prometheus is specific to BDI agents. Still, *independence* is thus limited. The process has no notion of agent organisation and focuses solely on *interactions* between the agents. This also limits the *separation of architecture domains*. A main feature are detailed and comprehensible guidelines that support the development steps, as well as support for validation, code generation, consistency checks, testing and debugging. These guidelines promote *extensibility* but it is unclear how the process can be adapted to different process lifecycles.

ADELFE has been specifically developed for the design of adaptive multi-agent systems (AMAS) with emergent functionality [14]. The methodology follows the Rational Unified Process (RUP) closely and uses UML and AUML, an extension of the UML meta-model with agent-specific concepts [15]. The method content for ADELFE is provided in the SPEM[1] format, making it *extensible* and reusable. It follows principles from the AMAS theory as well as classical object-oriented approaches. Adherence to the AMAS theory is also the main criteria when assessing the applicability of ADELFE for a specific system: it should have a certain complexity and should be open. Additionally, the development of algorithmic solutions to the core problems is an integral part of the process and therefore, the approach is mainly suitable when the algorithms are not known

[1] Software & Systems Process Engineering Metamodel (http://www.omg.org/spec/SPEM/2.0/), defined by the Object Management Group (OMG).

yet. This severely limits the methodology's *independence*. If an agent reaches a certain complexity in ADELFE, it is treated as a separate AMAS, thus providing a focus on *interaction* between agents and agent organisations. This also provides some separation of *architecture domains*, but the process does not provide guidelines on the separate, principled modelling of these domains.

ASPECS focuses on complex systems with an emphasis on holonic organisations [16] based on the PASSI methodology. A holon is here defined as "[...] self-similar structure composed of holons as sub-structures". The organisation into holons is captured in a meta-model that is used for the definition of the system structure. An important principle leveraged in ASPECS is the possibility of stepwise refinement of the holons. Like ADELFE, the methodology therefore has drawbacks w.r.t. *independence* and, in addition, relies on a specific meta-model. It is, however, extensible since the method content is available online. Both *separation of architecture domains* and a focus on *interactions* are ensured.

The *M*ulti*a*gent *S*ystems *E*ngineering methodology **MaSE** includes a development life cycle starting from the initial system specification and including implementation and deployment [12,13,17]. It has been applied in several research projects and has been lauded for its comprehensibility [11]. MaSE is *independent* of a specific agent architecture and can therefore be applied to heterogeneous systems [12]. A strength of the methodology is the way agent interactions and protocols are defined. Drawbacks are the complex description of concurrent tasks, the absence of models for the environment and agent behaviour, and missing specification tools for agent adaptivity [13,18]. In addition, the methodology was difficult to customise and organisational factors were not considered [19]. Based on this criticism, **O-MaSE** and "agentTool"[2] have been developed [19]. They provide a method engineering framework with which method fragments specified as SPEM[1] activities can be composed. The method content is based on a common meta-model and focuses mainly on analysis, design, and implementation. Organisations and the environment are now explicitly considered. *Extensibility* and *independence* are thus limited due to the specialised tool required and due to the meta-model. O-MaSE provides no overall system design activities, thus reducing the *separation of architecture domains.*

INGENIAS [20] aims at the development of organisational multi-agent systems and is the descendant of MESSAGE [21]. It uses meta-models to describe the relevant concepts in different aspects or "viewpoints" of a MAS, including organisation, agent, goals/tasks, interactions, and environment [20]. Relationships between the concepts for the different viewpoints are exploited to ensure consistency of the modelling. Meta-models are described in a specialised modelling language. The agents are based on BDI. INGENIAS is supported by specialised tools for modelling and process customisation. While this limits the *extensibility* and *independence* of the methodology, it offers full support for *separation of architecture domains* and for *interactions* between agents and agent organisations.

From the remarks above, it becomes clear that the other AOSE methodologies regarded do not fully support the particular set of requirements we identified.

[2] `http://agenttool.cis.ksu.edu/`

Table 1. Coverage of requirements for agent-oriented software engineering approaches

Requirement	Extensibility	Independence	Arch. Domains	Interaction
PosoMAS	Full	Full	Full	Full
OpenUP	Full	Full	No	No
Prometheus	Partial	No	Partial	Partial
ADELFE	Full	No	Partial	Full
ASPECS	Partial	No	Full	Full
O-MasE	Partial	Partial	Partial	Full
INGENIAS	No	No	Full	Full

The findings are summarised in Table 1. However, it must be noted that these requirements do not apply to all development situations. For some teams, it might be helpful to have a meta-model available or support by a dedicated tool. Others do not require support for agent organisations since the scale of the system under development is low or more complex organisational structures are not needed. In such situations, PosoMAS may not be an ideal candidate and one of the other methodologies may be better suited. It is thus important to consider the actual requirements of the development effort before choosing a process.

3 PosoMAS Practices

The practices for PosoMAS, compiled in a *practice library*, cover the disciplines requirements, architecture, and development. Testing and deployment are the focus of ongoing work (see, e.g., [22]) since both disciplines are very important in MAS and have not been dealt with sufficiently as of yet. The practices introduce techniques for the principled design of individual agents, organisations, their interactions, as well as the overall system and the environment. The categorisation of these techniques is an important aspect of the design of the process:

Agent Architecture. The design of the individual agents, separate for each distinct type of agent.

Agent Organisation. The specification of organisational paradigms that will structure the agent population at runtime.

Agent Interaction. The definition of interfaces and protocols used by agents to exchange information, delegate control, and change organisational structure.

System Architecture. The relationship between the different types of agents, the supporting infrastructure, external components, and the environment.

It is necessary to define common work products that can be used to exchange information between the activities and tasks specified for each of the areas. To structure these work products, respective SPEM domains (for work products) and disciplines (for tasks) have been introduced. The *agent system* and *individual agent* domains and disciplines complement the generic *architecture* domain and discipline. Agent interactions and agent organisations are captured in a respective domain and discipline as well.

3.1 Common Categories, Work Products, Roles, and Domains

The PosoMAS practices library introduces a number of work product slots, an additional role, changes in the interaction and responsibilities of the roles, and specialised domains. These elements allow the categorisation of artefacts within the development effort and provide a grouping of work products and tasks.

Work Products. They are used to exchange information between practices and capture the different architecture areas defined by PosoMAS. Work Product Slots are placeholders for concrete work products that allow interoperability between method content. Information about the agent architecture, e.g., is exchanged by a work product slot *[Agent Architecture]*. It serves as an abstraction of high-level artefacts that represent the documentation of the architecture of a single agent within a MAS.

Roles. A role fulfils certain tasks in the process, requires a certain skill set, and is usually assigned to one or more persons. To emphasise agile aspects, PosoMAS puts focus on the *Product Owner* who represents the client in the process. PosoMAS includes it in the requirements elicitation process and in the aspects that relate to the system environment. This changes the responsibilities of the *Analyst*, who is the liaison between the development team and the customer, since the customer is now more directly involved in the process. Likewise, the *Architect* works closely with the product owner during requirements elicitation.

Domains. PosoMAS introduces or extends four domains—specialised categories for the classification of work products—that relate to the different areas of the development effort. Work products can be related to *Agent Interaction* and *System Organisation*. In addition, the *Requirements* and *Architecture* domains from the practices library included with the EPF are supplemented with domains that contribute content to them.

3.2 Overview of PosoMAS Practices

As PosoMAS is targeted at open systems, the architectural tasks are aimed at providing standardisation, separation of concerns, and interoperability. The applicability to a wide range of target systems has also been a concern. Therefore, even though some content of the practices is specific to open self-organising multi-agent systems, they do not require the use of a specific meta-model or agent architecture. The practice library provides the following practices:

Goal-Driven Requirements Elicitation. Operationalises the technique for requirements elicitation based on KAOS [23] and the work of Cheng et al. [24]. It provides an iterative process composed of the tasks *Identify System Goals*, *Refine System Goals to Requirements*, *Mitigate Uncertainty Factors*, *Define System Limitations and Constraints*, and *Validate Requirements*. By applying these actions, the goal model is successively refined until a complete model of the system requirements is gained. Beside the system goal model, a conceptual domain model as well as a glossary of the domain are outputs of this practice.

The approach is ideally suited for adaptive systems since uncertainties and their mitigation can be directly modelled in the requirements. This allows the stakeholders to reason about countermeasures and identify risks early on. The practice is easily embedded in iterative-incremental processes. System goals can be elaborated in different iterations, with a preference to elaborate those first that have the greatest potential impact and risk. Guidelines detail the application of the practice in an agile environment and how to capture process support requirements.

Pattern-Driven MAS Design. Provides guidance to design a multi-agent system based on existing architectural, behavioural, and interaction patterns and reference architectures. These three types of patterns correspond to the system architecture, agent architecture, and agent interaction areas. A design conscientious of existing work enables reuse, avoids making mistakes twice, and allows tapping into the knowledge that has been created elsewhere for a cleaner, leaner, and more flexible design. An architectural pattern can be applied in the development of the system architecture, while more fine-grained patterns and protocols can be used to create agent architectures and define interactions between agents. The use of patterns also facilitates communication between stakeholders and makes the architecture and the implementation more comprehensible. The practice lists a wealth of published work containing patterns for the design of agents and MAS (e.g., [25]), including the FIPA Interaction Protocols Specification (`http://www.fipa.org/repository/ips.php3`).

Evolutionary Agent Design. Describes an approach to design agents and their architecture in an evolutionary way that enhances the design over time while requirements become clearer and development progresses. During the development process, the agent types, their capabilities and behaviour, their internal architecture, and their interactions become clearer as the requirements mature and the system design progresses towards a shippable build. To allow the product to mature this way, the design of the agents has to adapt to new knowledge continuously and become more specific by refinement when necessary and incorporating changes in the requirements or the system environment. The practice defines three tasks for the design of the different agent elements. These are tightly interwoven with tasks from *Pattern-driven MAS Design* in the PosoMAS lifecycle. Special guidance on the design of modular agents and the intricacies of message-based MAS is provided. A specialised UML profile helps the designer to identify agents, operations that are available through messaging, and to define elements of the infrastructure the development team relies on.

Agent System Design. Outlines how the system the agents are embedded in is designed and how the agents interact with it. A multi-agent system not only consists of autonomous agents but also incorporates a multitude of additional infrastructure, external actors, interfaces, hardware, and environmental factors. These can have a significant impact on the overall system design and should be regarded early on. The practice provides tasks to identify these factors and incorporate them in the design of the overall system. This includes the

identification and design of necessary interfaces between the agents and to external components in the system's environment as well as the identification of uncertainty factors in the environment. Additional guidance is provided with regard to the separation of concerns between system and agent level.

Agent Organisation Design. Describes the design of the organisation the agents will interact in. Multi-agent systems with many interacting agents require a form of structure or organisation imposed on the population. Sometimes, this structure is permanent, such as a hierarchy that determines the delegation of control and propagation of information, or transient, such as a coalition in which agents interact until a certain common goal is reached. The system designer has to decide which organisations are suitable for the system to reach the stated goals and implement mechanisms that allow the formation of these organisational structures at runtime. If this process is driven from within the system, "self-organisation" is present. This practice includes tasks, work products, and guidance that support the decision for a suitable system structure and the selection of a suitable self-organisation mechanism. If the system under development requires self-organisation, e.g., to be robust against agent failures or to adapt to a changing environment, these issues will have to be considered timely and thoroughly as the organisational structure and the algorithm to create it can have tremendous impact on the performance of the system. Introducing these concepts also influences the way the system is tested and deployed and has consequences for the operation of the deployed system. Possible system organisations and self-organisation approaches are, e.g., described in [26].

Model-Driven Observer Synthesis. Describes how observer implementations can be synthesized from constraints specified in the requirements documents as described in [4]. In adaptive systems, it is necessary to observe the system and react if the system enters an undesirable state or shows unwanted behaviour. For this purpose, feedback loops, operationalised as observer/controllers can be employed [27]. This practice describes an automatic transformation to observer/controller implementations from constraints defined during requirements analysis. A prerequisite of this practice is that constraints have been captured during requirements analysis. Ideally, these are expressed as OCL constraints that define the correct states of the system. If *Define System Limitations and Constraints* from the practice *Goal-driven Requirements Elicitation* is performed, constraints and a domain model should be available. At the same time, this ensures that a domain model containing the elements the constraints are defined on is available. Constraints can also be defined in a specialised document separate from the requirements model. The process can be repeated after the requirements or the domain model have changed, according to a model-driven design (MDD) approach. Changed parts of the system models and implementation will be re-generated while existing models and code are preserved.

Trust-Based Interaction Design. Guides the design of interactions in open systems that can be evaluated with trust models and agent decisions that use trust values to make the system more robust and efficient. Trust-based

interaction design enables the agents in the system to determine and select trustworthy interaction partners with a high likelihood of successful completion of an interaction. The added overhead of using trust is often justified if the interactions have a high risk or a high impact. Trust helps make the system more robust against unintentional and malevolent interaction behaviour and can even enable more efficient problem solving. To be effective, trust values need to be calculated using a trust model (see, e.g., [28]) that allows the quantification of an interaction's outcome. It is also helpful to define the intended outcomes with an implicit or explicit contract. The concept of an interaction can be regarded very generally. Not only are communications with other agents an interaction but also querying of sensors, the use of environmental data, and others. All these interactions are sources of uncertainties that can be mitigated by trust. The practice supports the design of the trust model, the decision making process of agents based on trust values, the design of an infrastructure to measure trust values, and the design of a repuatation system.

Each practice is defined by an appropriate guidance in EPF that states the purpose of the practice, gives a description, and provides a brief instruction on how the elements of the practice relate to each other and in which order they should be read. The practice usually references a roadmap (another special type of guidance) for the adoption of the practice, a list of key concepts and white papers, and a set of guidances. A practice also takes one or several work products (or work product slots) as inputs and outputs. These work products are automatically derived from the respective relationships of the tasks. If the practices are combined into a process, the outputs of the practices can be used to instantiate the work product slots denoting the inputs of the other practices. The *Conceptual Domain Model* can, e.g., be used to fill the [Multi-Agent System Architecure] in early iterations of the process.

The detailed practice descriptions and the models for use in EPF are available at `http://posomas.isse.de`. We thus provide a repository for method content and make reusable assets available for combination with method content from other processes, fulfilling the appeal of the IEEE FIPA Design Process Documentation and Fragmentation Working Group and many authors (e.g., [29, 30]).

4 The PosoMAS Life Cycle

The process life cycle determines the progression of a product under development in the context of the process, the stakeholders, and the environment. A well-defined life cycle provides guidance w.r.t. the order of activities the development team has to perform, the project milestones and deliverables, and the progress of the product. The advancement of a product development effort can thus be measured based on the planned and the actual progress within the life cycle. A methodology is created by embedding the activities and tasks defined in the practices into a life cycle. The structure the life cycle provides is often defined by phases (e.g., inception, elaboration, construction, and transition in the OpenUP as described below) that are executed sequentially. Each phase

addresses different needs within the project and in general, a shift away from requirements, towards design and then implementation and testing is evident.

4.1 The Open Unified Process as a Method Template

The PosoMAS practices are embedded in the risk-value life cycle of the *Open Unified Process* (OpenUP) [31, 32]. It promotes an approach in which the most risky requirements and those that provide the greatest value are tackled first in an iterative-incremental way, in which each phase consists of a number of iterations. It is a lean, agile, process and towards an extensible process framework that provides a starting point for customisations and extensions.

The technical practices described in the OpenUP practice library deal with all disciplines of the development process. In general, they are described on a very high level of abstraction. Therefore, most of them are replaced by more specific practices defined by PosoMAS. For example, the practice *Evolutionary Design* is superseded by PosoMAS' *Evolutionary Agent Design*. However, the proposed process borrows a number of practices from testing and deployment as these areas are still under active development. The EPF practice library also contains a "core" area in which common elements are defined, including categories, roles, work products (and work product slots), and guidance, some of which are refined by the practices in the PosoMAS practices library.

4.2 The PosoMAS Life Cycle and Work Breakdown Structure

PosoMAS adds most of its method content in the design activities and replaces use cases with system goals as the main model to capture requirements. It also adopts the OpenUP project and iteration life cycle by incorporating the EPF practices *Risk-Value Life Cycle* and *Iterative Development* which divide the work in PosoMAS in four phases. In each phase, specialised activities are applied to accommodate open self-organising multi-agent systems. They are all specified in detail by activity diagrams such as the one in Figure 1.

The **inception phase** is often iterated only once and lays the foundational work for the project. The development team, the product owner, and the stakeholders have to come to an agreement about the scope of the project, including the features of the system and the final quality standards (task *Develop Technical Vision*, practice *Shared Vision*). For this purpose, extensive requirements elicitation is performed. PosoMAS uses goal-driven requirements elicitation from the practice of the same name. The requirements and the shared vision are also used to *Agree on a Technical Approach* (includes the task *Envision Architecture*, practice *Evolutionary Architecture*).

Notably, the activity *Plan and Manage Iteration* addresses *Prepare Environment* and *Project Process Tailoring* during inception. As in later phases, its outputs are an *Iteration Plan* and a *Work Items List* that describe the timetable and break down of work packages, as well as a *Risk List* containing critical points that need to be addressed. The inception phase ends with a *Life Cycle Objectives*

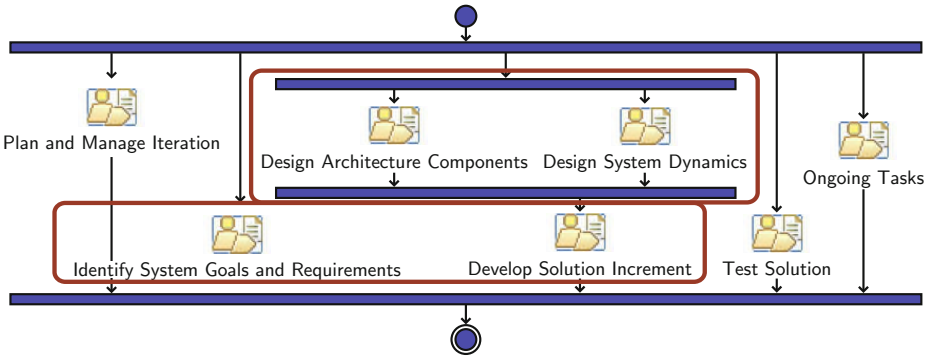

Fig. 1. Overview of the elaboration phase of the PosoMAS. Red frames indicate original PosoMAS content.

Milestone that determines the project scope and the objectives the project has to fulfil at the end of the inception phase.

The **elaboration phase** puts the focus of the development team on the design of the software and the realisation of the requirements. At the same time, first implemented features generate value for the customer and are the basis for further elaboration of the requirements. The activity diagram in Figure 1 shows the most important activities. System design activities are now added. Early implementation and testing are also performed, along with change management.

Design activities include *Design Architecture Components*, *Design System Dynamics*, and *Develop Solution Increment*. Requirements are selected and design and subsequent implementation are then performed to develop an increment that provides value to the customer and reduces the risk inherent in the project. *Design Architecture Components* deals with the static parts of the system design and the trust infrastructure and includes sub-activities for system architecture, agent architecture, and trust-based interaction design. It also includes a task from *Model-driven Observer Synthesis* for the definition of the observation model. The use of patterns and re-usable architectural elements is promoted by incorporating *Pattern-driven MAS Design*.

Design System Dynamics deals with the behaviour of the agents, their interactions, and agent organisations. Capabilities of the agents are identified and their behaviour is specified. The interactions between the agents are designed and interaction patterns and protocols are applied if possible. A suitable system organisation is selected and a self-organisation algorithm is specified if necessary. *Develop Solution Increment* can be performed after these design activities have been completed. Tests are carried out in *Test Solution* and if all tests pass, the code is integrated and a build is created. *Ongoing Tasks* deals with the submission and integration of change requests.

The elaboration phase ends with the *Life Cycle Architecture Milestone* that signifies that the most important aspects of the system, agent, and organisational architecture are completed. The most risky requirements have been tackled and appropriate solutions have been incorporated into the design.

The **construction phase** marks a shift from design and requirements elicitation towards implementation and preparation for an eventual release. While the overall structure of an iteration in this phase is similar to prior ones, the overall design has become stable and most design activities are no longer performed. Release and documentation activities are newly introduced.The phase ends with the *Initial Operational Capability Milestone*, an extended prototype that is usable as a standalone product. Testing is mostly finished and a preliminary product documentation as well as plans for deployment are available.

The final iterations of the process are part of the **transition phase** in which development is wrapped up and a final release is created and deployed. Change requests no longer result in new requirements or changes in the design but have to be realised on the code level. Product documentation and training documents are finished and product training starts. Release preparations are completed and the final release is deployed. The transition phase ends with the *Product Release Milestone* including the accepted final product, complete training and documentation, as well as successful deployment.

5 Evaluation and Comparison to Other AOSE Processes

The validation of a software engineering process is difficult from a methodical point of view. Ideally, the process is tested in a productive environment for the creation of a software product with an experienced team of software engineers and developers who can compare the effort to previous experiences with other methodologies. Such an approach, however, is not feasible in the scope in which AOSE methodologies are created at the moment. Instead, we rely on qualitative evaluation and validation criteria. Tran et al. [8,11] have introduced a catalogue of criteria that are used in Table 2 to show the characteristics of PosoMAS and to compare it to other approaches. It is important to note that the table only captures if a process has explicit supporting content for a certain criterion. It is, e.g., possible to build proactive systems with PosoMAS even though the process does not include specific support for them. The process website contains a detailed description of the criteria and comparisons under different aspects and for additional methodologies at `http://posomas.isse.de`.

The basis for these evaluations are simulated development efforts for two case studies: a self-organising emergency response system and a power management system. The former system is highly connected and includes sensors, information retrieval and distribution, as well as a pronounced human component. The power management case study on the other hand puts the focus on self-organisation and self-optimisation in a fully autonomous system. It is available as an example run at `http://posomas.isse.de` along with a detailed description and a selection of artefacts. This diversity allows us to demonstrate that PosoMAS is applicable to a wide range of open multi-agent systems if tailored appropriately.

The development of PosoMAS and the accompanying validation provided a number of lessons that have been integrated in the process and its documentation. First and foremost, the distinction of *architecture areas* is vital for the

Table 2. Characteristics of PosoMAS, O-MasE, Prometheus, and ASPECS based on [8–11, 16, 19]. More details on criteria and values on `http://posomas.isse.de`.

Criteria	**PosoMAS**	**O-MasE**	**Prometheus**	**ASPECS**
Process-Related Criteria				
Development lifecycle	Iterative-incremental risk-value life cycle	Depends on base process	Iterative across all phases	Iterative-incremental life cycle
Coverage of the lifecycle	Conceptualisation, Analysis, Design, (Test, Deployment, Management)	Analysis, Design	Analysis, Design	Analysis, Design, Test, Deployment
Development perspectives	Hybrid	Top-Down	Bottom-Up	Top-Down
Application Domain	Any	Any	Any	Any
Size of MAS	Not Specified	Not Specified	Not Specified	Not Specified
Agent paradigm	Heterogeneous	Heterogeneous	BDI	Holonic
Model Validation	Yes	Consistency	Consistency and completeness	No
Refinability	Yes	Yes	Yes	Yes
Approach towards MAS development	Object-Oriented, Non-Role-Based	Object-Oriented, Role-Based, Goal-Oriented	Object-Oriented, Non-Role-Based	Role-Based, Knowledge Engineering
Meta-model based	No	Yes	No	Yes
Model-Related Criteria				
Syntax and Semantics	Medium	High	High	High
Model transformations	Yes	Yes	Yes	Yes
Consistency	Yes	Yes	Yes	Yes
Modularity	Yes	Yes	Yes	Yes
Abstraction	Yes	Yes	Yes	Yes
Autonomy	Yes	Yes	Yes	Yes
Adaptability	No	Yes	No	Yes
Cooperation	Yes	Yes	Yes	Yes
Inferential capability	No	Yes	Yes	No
Communication	Yes	Yes	Yes	Yes
Reactivity	Yes	Yes	Yes	Yes
Proactivity	No	Yes	Yes	Yes
Concurrency	No	Yes	No	No
Model Reuse	Yes	Yes	Yes	Yes
Supportive Feature Criteria				
Software and methodological support	Yes	Yes	Yes	Yes
Open systems and scalability	Yes	No	No	Yes
Dynamic structure	Yes	Yes	No	Yes
Performance and robustness	Yes	No	Yes	Yes
Support for conventional objects	Yes	No	Yes	Yes
Support for self-interested agents	Yes	No	No	Yes
Support for ontologies	No	No	No	Yes

creation of a modular, flexible design. Many of the problems with the initial system design in early iterations were caused by a misunderstandings about which parts of the design were on the agent level, which on the system level and in the environment, and which are part of the organisation design. These areas have thus been discriminated more thoroughly and according tasks and guidance has been disentangled. Second, the concept of *scope* and thus of the system boundaries has been overhauled and extended from the guidance provided by the OpenUP or existing AOSE processes. Essentially, everything outside the scope the system has to interact with, can not simply be ignored but assumptions must be captured and the environment has to be modelled accordingly. Finally, a specialised UML profile containing stereotypes for agents, methods that are part of an agents interface, and external components was introduced to mark specific concepts in the agent and system models.

6 Discussion and Future Work

This paper introduced PosoMAS, a novel agent-oriented software engineering process for the class of large-scale open self-organising systems. It is based on a risk-value lifecycle and incorporates practices both for agile development and for the principled design and implementation of self-organising systems. It has been validated in two case studies and compared with a number of other agent-oriented processes.

The level of abstraction differs tremendously between different processes. While the OpenUP is very abstract, without domain- or problem-specific guidance, Prometheus, ASPECS and other AOSE-processes are very concrete and prescribe solution approaches, techniques, and models in great detail. The latter approach excels when a system fits the assumptions made by giving much more hands-on support. However, it is rare that a product fits the assumptions perfectly. PosoMAS tries to find a middle ground between these extremes by providing guidance without forcing adherence to a special paradigm and by formulating method content in a way that lends itself to process customisation and tailoring. The comparisons in Table 1 and Table 2 can provide indications of the strength and weaknesses of the different processes.

Most processes impose a certain way of thinking about the system under construction. Prometheus enforces the use of BDI-agents, O-MaSE puts the focus on organisations, and ASPECS forces developers to think in terms of ontologies and holarchies. PosoMAS has been designed to be independent of most of these factors but still contains elements that favour certain solutions, e.g., using the Observer/Controller architectural pattern as the basis for adaptation. When choosing a process, the development team has to make sure that the perspective taken by the process is compatible with the product.

Future work includes the creation and integration of additional method content, especially w.r.t. deployment and testing of self-organising systems. Furthermore, the method content will be combined with the principles of the Scrum methodology to yield a truly agile process. These efforts will be accompanied

by evaluations and refinement of the method content. Our hope, however, is that by making PosoMAS and all method content available as a repository at `http://posomas.isse.de` both in browsable form and as EPF source code, other researchers and practitioners will start using the practices and the framework they provide to adapt the process, create new methodologies, and enrich the content with their own ideas and concepts[3].

Acknowledgements. This research is partly sponsored by the research unit *OC-Trust* (FOR 1085) of the German Research Foundation. The authors thank Julian Kienberger for his input on the comparison of existing AOSE approaches.

References

1. Steghöfer, J.P., Anders, G., Siefert, F., Reif, W.: A system of systems approach to the evolutionary transformation of power management systems. In: Proc. of INFORMATIK 2013 – Workshop on Smart Grids. LNI, vol. P-220. Bonner Köllen Verlag (2013)
2. Bernard, Y., Klejnowski, L., Müller-Schloer, C., Pitt, J., Schaumeier, J.: Enduring institutions and self-organising trust-adaptive systems for an open grid computing infrastructure. In: Proc. of the 2012 Sixth IEEE Int. Conf. on Self-Adaptive and Self-Organizing Systems Workshop (SASOW), pp. 47–52. IEEE (2012)
3. Sudeikat, J., Steghöfer, J.P., Seebach, H., Reif, W., Renz, W., Preisler, T., Salchow, P.: On the combination of top-down and bottom-up methodologies for the design of coordination mechanisms in self-organising systems. Information and Software Technology 54(6), 593–607 (2012)
4. Eberhardinger, B., Steghöfer, J.P., Nafz, F., Reif, W.: Model-driven Synthesis of Monitoring Infrastructure for Reliable Adaptive Multi-Agent Systems. In: Proc. of the 24th IEEE Int. Symposium on Software Reliability Engineering (ISSRE 2013). IEEE Computer Society, Washington, D.C (2013)
5. Steghöfer, J.P., Behrmann, P., Anders, G., Siefert, F., Reif, W.: HiSPADA: Self-Organising Hierarchies for Large-Scale Multi-Agent Systems. In: 9th Int. Conf. on Autonomic and Autonomous Systems, ICAS 2013, Lisbon, Portugal, pp. 71–76. IARIA (March 2013)
6. Kroll, P., Kruchten, P.: The Rational Unified Process Made Easy—A Practitioner's Guide to the RUP. Addison-Wesley Professional (2003)
7. Henderson-Sellers, B., Gonzalez-Perez, C., Ralyte, J.: Comparison of method chunks and method fragments for situational method engineering. In: 19th Australian Conf. on Software Engineering, ASWEC 2008, pp. 479–488 (2008)
8. Tran, Q.N.N., Low, G.C.: Comparison of ten agent-oriented methodologies. In: Agent-oriented Methodologies, pp. 341–367. Idea Group, Hershey (2005)
9. Cossentino, M., Hilaire, V., Molesini, A., Seidita, V.: Handbook on Agent-Oriented Design Processes. Springer, Heidelberg (2014)
10. Padgham, L., Winikoff, M.: Developing Intelligent Agent Systems. John Wiley & Sons, Ltd. (2005)

[3] All content is made available under the Creative Commons–Attribution-ShareAlike License v3.0 (`http://creativecommons.org/licenses/by-sa/3.0/`).

11. Tran, Q.-N.N., Low, G., Williams, M.-A.: A preliminary comparative feature analysis of multi-agent systems development methodologies. In: Bresciani, P., Giorgini, P., Henderson-Sellers, B., Low, G., Winikoff, M. (eds.) AOIS 2004. LNCS (LNAI), vol. 3508, pp. 157–168. Springer, Heidelberg (2005)
12. Al-Hashel, E., Balachandran, B.M., Sharma, D.: A Comparison of Three Agent-Oriented Software Development Methodologies: ROADMAP, Prometheus, and MaSE. In: Apolloni, B., Howlett, R.J., Jain, L. (eds.) KES 2007, Part III. LNCS (LNAI), vol. 4694, pp. 909–916. Springer, Heidelberg (2007)
13. Dam, K.H., Winikoff, M.: Comparing agent-oriented methodologies. In: Giorgini, P., Henderson-Sellers, B., Winikoff, M. (eds.) AOIS 2003. LNCS (LNAI), vol. 3030, pp. 78–93. Springer, Heidelberg (2004)
14. Bernon, C., Gleizes, M.-P., Peyruqueou, S., Picard, G.: ADELFE: A methodology for adaptive multi-agent systems engineering. In: Petta, P., Tolksdorf, R., Zambonelli, F. (eds.) ESAW 2002. LNCS (LNAI), vol. 2577, pp. 156–169. Springer, Heidelberg (2003)
15. Bauer, B., Müller, J.P., Odell, J.: Agent UML: A formalism for specifying multi-agent software systems. In: Ciancarini, P., Wooldridge, M.J. (eds.) AOSE 2000. LNCS, vol. 1957, pp. 91–103. Springer, Heidelberg (2001)
16. Cossentino, M., Gaud, N., Hilaire, V., Galland, S., Koukam, A.: ASPECS: an agent-oriented software process for engineering complex systems. Autonomous Agents and Multi-Agent Systems 20(2), 260–304 (2010)
17. DeLoach, S.A., Wood, M.F., Sparkman, C.H.: Multiagent Systems Engineering. IJSEKE 11(3), 231–258 (2001)
18. Abdelaziz, T., Elammari, M., Unland, R.: A Framework for the Evaluation of Agent-Oriented Methodologies. In: 4th Int. Conf. on Innovations in Information Technology, IIT 2007, pp. 491–495 (November 2007)
19. DeLoach, S.A., Garcia-Ojeda, J.C.: O-MaSE – a customisable approach to designing and building complex, adaptive multi-agent systems. IJAOSE 4(3), 244–280 (2010)
20. Pavón, J., Gómez-Sanz, J.: Agent oriented software engineering with INGENIAS. In: Mařík, V., Müller, J.P., Pěchouček, M. (eds.) CEEMAS 2003. LNCS (LNAI), vol. 2691, pp. 394–403. Springer, Heidelberg (2003)
21. Caire, G., Coulier, W., Garijo, F., Gomez, J., Pavon, J., Leal, F., Chainho, P., Kearney, P., Stark, J., Evans, R., Massonet, P.: Agent oriented analysis using message/UML. In: Wooldridge, M.J., Weiß, G., Ciancarini, P. (eds.) AOSE 2001. LNCS, vol. 2222, pp. 119–135. Springer, Heidelberg (2002)
22. Eberhardinger, B., Seebach, H., Knapp, A., Reif, W.: Towards testing self-organizing, adaptive systems. In: Merayo, M.G., de Oca, E.M. (eds.) ICTSS 2014. LNCS, vol. 8763, pp. 180–185. Springer, Heidelberg (2014)
23. van Lamsweerde, A., Letier, E.: From object orientation to goal orientation: A paradigm shift for requirements engineering. In: Wirsing, M., Knapp, A., Balsamo, S. (eds.) RISSEF 2002. LNCS, vol. 2941, pp. 325–340. Springer, Heidelberg (2004)
24. Cheng, B., Sawyer, P., Bencomo, N., Whittle, J.: A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty. In: Schürr, A., Selic, B. (eds.) MODELS 2009. LNCS, vol. 5795, pp. 468–483. Springer, Heidelberg (2009)
25. Ramirez, A.J., Cheng, B.H.C.: Design patterns for developing dynamically adaptive systems. In: Proc. of the Workshop on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2010, pp. 49–58. ACM, New York (2010)
26. Horling, B., Lesser, V.: A survey of multi-agent organizational paradigms. The Knowledge Engineering Review 19(04), 281–316 (2004)

27. Richter, U., Mnif, M., Branke, J., Müller-Schloer, C., Schmeck, H.: Towards a generic observer/controller architecture for Organic Computing. In: 36. Jahrestagung der GI. LNI, vol. 93, pp. 112–119. GI (2006)
28. Pinyol, I., Sabater-Mir, J.: Computational trust and reputation models for open multi-agent systems: a review. Artificial Intelligence Review 40(1), 1–25 (2013)
29. Seidita, V., Cossentino, M., Gaglio, S.: A repository of fragments for agent systems design. In: Proc. of the Workshop on Objects and Agents (WOA 2006), Catania, Italy, pp. 130–137 (September 2006)
30. Cossentino, M., Gleizes, M.-P., Molesini, A., Omicini, A.: Processes engineering and AOSE. In: Gleizes, M.-P., Gomez-Sanz, J.J. (eds.) AOSE 2009. LNCS, vol. 6038, pp. 191–212. Springer, Heidelberg (2011)
31. Eclipse Foundation: Openup (2013), `http://epf.eclipse.org/wikis/openup/` (accessed September 2, 2013)
32. Gustafsson, B.: Openup – the best of two worlds. Methods & Tools (2008)