# Chapter 7
# The Landscape of Agent-Oriented Methodologies

**Arnon Sturm and Onn Shehory**

**Abstract** Agent-based systems have evolved during the last two decades. To support the development of such systems, agent-oriented methodologies have emerged. In general, most of the methodologies have originated from two major research domains, namely software engineering and artificial intelligence, and were adjusted to address the agent abstraction. It seems that many of the methodologies share a common basis, an observation that calls for unification and for standardization. In this chapter, we survey existing agent-oriented methodologies and describe the support for agent-based concepts required in such methodologies. We then analyze the most influential agent-oriented methodologies in light of the required agent-based concepts as well as other criteria. We also examine alternatives such as methodology integration and the support for developing a tailored agent-oriented methodology. The main concern that arises from the survey and the analysis is the lack of evaluation of agent-based methodologies, which may have negatively affected, at least in part, the adoption of these methodologies for developing agent-based systems. We also discuss the need to further extend the methodologies to support the entire lifecycle.

**Keywords** Software Development • Agent-Oriented Methodologies • Evaluation • Comparison

A. Sturm (✉)
Department of Information Systems Engineering, Ben-Gurion University of the Negev, Beer-Sheva, Israel
e-mail: sturm@bgu.ac.il

O. Shehory
IBM – Haifa Research Lab, Haifa, Israel
e-mail: onn@il.ibm.com

O. Shehory and A. Sturm (eds.), *Agent-Oriented Software Engineering*,
DOI 10.1007/978-3-642-54432-3_7, © Springer-Verlag Berlin Heidelberg 2014

## 1 Introduction

During the last 15 years, many methodologies for developing agent-based systems have been developed. When referring to a methodology, we follow the definition of [1], according to which a methodology should provide the following: a full lifecycle process; a comprehensive set of concepts and models; a full set of techniques (rules, guidelines, heuristics); a fully delineated set of deliverables; a modeling language; a set of metrics; quality assurance; coding (and other) standards; reuse advice; and guidelines for project management. Naturally, Agent-Oriented Software Engineering (AOSE) methodologies adopt general software engineering concepts. However, their coverage of activities required from a comprehensive methodology is partial, as they mainly focus on technical issues and not on managerial ones.

Agent-oriented methodologies can be classified into two major classes: general-purpose methodologies and domain-specific methodologies. In this chapter we refer to the former. Methodologies of that kind have emerged from several disciplines, but mainly from the classical software engineering, in times influenced by the knowledge engineering stream. When referring to software engineering, these methodologies emphasize the way in which agent-based systems should be constructed following software engineering principles (e.g., expressiveness, accessibility, and reuse). When referring to knowledge engineering, these methodologies emphasize implementation mechanisms of agents within the system (e.g., cooperation, reasoning, negotiation) in terms of the knowledge perspective. Table 7.1 lists most of the agent-oriented methodologies that were developed during the last two decades. In that table, the methodologies are classified according to their domains of origin.

This chapter aims to explore the plethora of AOSE methodologies and the way these were evaluated. This should shed light on research and practice in this area and present challenges in the field of AOSE methodologies.

The chapter is organized as follows. In Sect. 2, we introduce and define a set of evaluation criteria. These are then used in Sect. 3, where we shortly introduce the main methodologies and evaluate them. In Sect. 4, we discuss alternative approaches for AOSE methodologies. Section 5 concludes, presenting major challenges in the area of AOSE methodologies.

## 2 Criteria for Examining AOSE Methodologies

Several techniques for evaluating a methodology or comparing it to alternatives exist. These include the following:

- Feature analysis, in which a set of features is determined before the evaluation takes place followed by subjective grading of the features for each methodology.
- Survey, in which a methodology is evaluated by an individual, usually following a questionnaire.

**Table 7.1** AOSE methodologies

| Methodology name | Domain of origin | Methodology name | Domain of origin |
|---|---|---|---|
| AAII [2, 3] | AI-KE | MAS-CommonKADS [4] | SE + AI-KE |
| ADELFE [5] | SE | MASSIVE [6] | SE |
| ADEM [7] | SE | MESSAGE [8] | SE |
| ADEPT [9–11] | AI-KE | Nemo [12] | SE + AI-KE |
| AO [13] | SE | ODAC [14] | SE |
| AOR [15] | SE | OPEN for MAS [16] | SE |
| Cassiopeia [17] | SE | PASSI [18] | SE + AI-KE |
| CoMoMas [19] | SE + AI-KE | Prometheus [20] | SE + AI-KE |
| DESIRE [21–23] | AI-KE | Roadmap [24] | SE + AI-KE |
| FAF [25–29] | SE | SADDE [30] | SE |
| GAIA [31–33] | SE | SODA [34] | SE |
| INGENIAS [35] | SE | Styx [36] | SE |
| MASD [37] | SE + AI-KE | Tropos [38] | SE + AI-KE |
| MaSE [39] | SE | | |

*SE* Software Engineering, *AI-KE* Artificial Intelligence and Knowledge Engineering

- Case study, in which the strengths of the methodology are demonstrated on a case study. Usually, case studies are of limited extent and are within the context of research projects.
- Field experiment, in which a methodology is examined in a real world environment, yet with some kind of control.
- Lab experiment, in which a methodology is examined in an artificial setting.
- Qualitative approach, in which the methodology is examined for understanding phenomena that occur during its use. The approach may use techniques such as observations, interviews, and think aloud techniques.

Many of the aforementioned evaluation techniques are difficult to apply. Therefore, they are rarely used. Thus, in the area of AOSE methodologies, evaluation is performed using case studies for specific methodologies, and comparison is done using the feature analysis approach. The latter is also used in this chapter. Multiple research efforts were allocated to feature-based evaluation frameworks for agent-oriented methodologies. In [40], the authors set a list of questions that a methodology should address. In [41], the authors suggest a framework for evaluating agent-oriented methodologies referring to the expressiveness of the methodologies. In [42], the authors perform an evaluation of the modeling part within a methodology. In [43] and [44], a comparison of several agent-oriented methodologies is presented. In [45], [46], and [47], the authors also present alternative frameworks for evaluating agent-oriented methodologies; the frameworks suggest different sets of criteria with partial overlaps. Due to the proliferation of evaluation frameworks, in [48] the authors performed some meta-analysis and proposed a profile-based approach to examiningAOSE methodologies.

In this chapter, we follow the set of characteristics that was adopted by [48] and explain each of these.

## 2.1 Concepts and Properties

As mentioned within the first chapter of this book, agent-based systems are unique in several characteristics. These are elaborated in the following:

1. **Autonomy** is the ability of an agent to operate without supervision.
2. **Reactiveness** is the ability of an agent to respond in a timely manner to changes in the environment.
3. **Proactiveness** is the ability of an agent to pursue new goals.
4. **Mental notions** is the ability if an agent to refer to metal attitudes such as a belief (which is a fact about the world), a desire (which is a fact an agent would prefer that it be true), and an intention (which is a fact that represents the way of realizing a desire).
5. **Organization** is a group of agents working together to achieve a common purpose. An organization consists of roles that characterize the agents, which are members of the organization.
6. **Protocol** is an ordered set of messages that together define the admissible patterns of a particular type of interaction between entities.

## 2.2 Modeling and Notations

When examining a methodology that includes modeling capabilities, one should look at the following issues:

1. **Analyzability** is a capability to check the internal consistency or implications of models.
2. **Complexity management** (abstraction) is an ability to deal with various levels of abstraction (i.e., various levels of detail).
3. **Expressiveness** is a capability of presenting system concepts that refer to: the structure of the system; the knowledge encapsulated within the system; the system's ontology; the data flow within the system; the control flow within the system; the concurrent activities within the system (and the agents); the resource constraints within the system (e.g., time, CPU, and memory); the system's physical architecture; the agents' mobility; the interaction of the system with external systems; and the user interface definitions.
4. **Accessibility** is the ability that refers to the ease, or the simplicity, of understanding and using a method.

### 2.3 Process

A development process is a series of actions, changes, and functions that, when performed, result in a working computerized system. In particular, we refer to the **lifecycle coverage** and the development stages that are supported by the methodology.

### 2.4 Pragmatics

A methodology requires support for using it. Thus, from a pragmatic viewpoint it is beneficial to examine the extent to which the methodology addresses the following.

1. **Resources:** What resources are available in order to support the methodology? Is a textbook available? Are users' groups established? Are training and consulting offered by the vendor and/or third parties? In addition, are automated tools (CASE tools) available in support of the methodology (e.g., graphical editors, code generators, and checkers)? This issue should be examined in order to enable a project/organization aiming at adopting a methodology to check the resources (in terms of training and budget) required and the alternatives for acquiring these.
2. **Domain applicability:** Is the use of the methodology suitable for a particular application domain (e.g., real-time and information systems)?
3. **Scalability:** Can the methodology, or subsets thereof, be used to handle various application sizes? For example, can it provide a lightweight version for simpler problems? This issue should be examined to check whether the methodology is appropriate for handling the intended scale of applications within the project/organization.

## 3 Analysis of Existing AOSE Methodologies

In this section, we review several methodologies selected from the list above and discuss their evolution and their characteristics. The selection of these methodologies was motivated by the continuous flow of publications as well as their impact on the AOSE methodologies field. As the purpose of this chapter is to provide an overview of the field, we focus on evaluation criteria that are accessible to a wide readership and can be objectively measured.

### 3.1 GAIA

GAIA is an agent-oriented methodology that is not coupled to a specific programming language nor deals with implementation issues. GAIA [31, 33] provides a

set of models that are used at the analysis and design stages of the multi-agent system development and evolve over that process. Following the (plain vanilla) GAIA guidelines, the analysis of an agent-based system results in an environmental model, a preliminary role model, a preliminary interaction model, and a set of organizational rules.

- The **environmental** model is intended as an abstract, computational representation of the environment in which the multi-agent system will be situated.
- The **preliminary role** model is used for identifying the basic skills required by the organization and contains only those roles, possibly not completely defined, that can be identified without committing to the imposition of a specific organizational structure. Also, the notion of roles, at this stage, is abstracted from any mapping into agents.
- The **preliminary interaction** model specifies the basic interactions required to accomplish the preliminary roles. This model must abstract away from the organizational structure and can be left incomplete.
- The **organizational rules** that should be respected and enforced. These rules express constraints on the execution activities of roles and protocols of the organization.

In the architectural design stage, there are two main activities:

- The **definition of the system's organizational structure** in terms of its topology and control regime. This activity, which could also exploit catalogue organizational patterns, involves considering: (1) the organizational efficiency, (2) the real-world organization in which the multi-agent system is situated, and (3) the need to enforce the organizational rules.
- The **completion of the preliminary role** and **interaction models**.

Finally, in the detailed design stage the following activities are executed:

- The **definition of the agent model**. This identifies the agent classes that will make up the system and the *agent instances* that will be instantiated from these classes.
- The **definition of the services model**. This identifies the main services intended as coherent blocks of activity, in which agents will engage, that are required to realize the agent's roles, and their properties.

Note that GAIA has many extensions and elaborations such as [24, 49–53]. These include CASE tool development, additional models, moving towards implementation, case studies, etc.

Examining GAIA in light of the evaluation criteria, we found out that it supports well all multi-agent concepts except for the mental notion (i.e., BDI). With respect to the modeling and notation aspect, GAIA requires further attention, mainly in analyzing the specification and in its scalability. As for the development lifecycle coverage, although many extensions have been made to GAIA, further adjustments are still required. From the pragmatic point of view, as there is no coordinating effort of developing GAIA, its resources are limited.

## *3.2 INGENIAS*

INGENIAS is a methodology for the development of multi-agent systems, which is based on the well-known, well-established software development process, the unified process. It is based on a definition of a set of meta-models that describe the elements that form a multi-agent system from several viewpoints, and that allow to define a specification language for MAS. There are five viewpoints within INGENIAS: agent, interactions, organization, environment, and goals/tasks. In the following, we elaborate on these views.

- The **organization** view consists of agents and their groups from the structural point of view and the goals and workflows they should execute from the behavioral point of view.
- The **environment** view consists of the elements that surround the systems and the various stimuli, as well as the system resources.
- The **tasks/goals** view consists of a description of the agent mental states and the way they change over time, the consequence of executing a task with respect to the mental state of an agent, and how to achieve goals.
- The **agent** view consists of the primitives that describe a single agent. It can be used to define the capabilities of an agent or its mental state.
- The **interaction** view consists of the description of two or more agents interacting.

INGENIAS uses the Meta-Object Facility (MOF) standard to describe the five-view metamodels, to enable easy change when required.

As for the development process, INGENIAS adopts the unified process concepts and provides more than 70 activities to be performed during the development process of a multi-agent system. In general, INGENIAS has two main workflows: the analysis and the design. During the analysis workflow, it is expected to generate use cases and identify their actors, sketch system architecture with an organization model, and generate environment models. Next, there is a need to refine use cases and the interactions associated with them, develop agent models that detail elements of the system architecture, describe workflows and tasks in organization models, and refine the environment model. During the design workflow, the first task is to generate a prototype (out of the analysis results); then, refinements of the workflows should be introduced; following, it is necessary to specify the interaction models, to model tasks and goals, and to define the agent models. Finally, the social relationships that regulate organizational behavior should be outlined. According to INGENIAS, the implementation of MAS follows the Model-Driven Engineering (MDE) approach. Also, note that INGENIAS supports an iterative approach in which the workflows and activities are repeated during the development lifecycle.

Similarly to GAIA, INGENIAS supports well all multi-agent concepts except for the mental notion (i.e., BDI). With respect to the modeling and notation aspect, it seems that using INGENIAS requires more training. INGENIAS covers most of the development lifecycle. From a pragmatic point of view, INGENIAS was applied in

many contexts and has a supporting IDE. A detailed description of INGENIAS and its evolution is described in Chap. 10.

## 3.3 MaSE

Multi-agent Systems Engineering (MaSE) is a general-purpose methodology for developing heterogeneous multi-agent systems [39, 54]. MaSE uses a number of graphical models to describe system goals, behaviors, agent types, and agent communication interfaces. It uses most of the Unified Modeling Language (UML) diagrams and makes some enhancements to adjust them to the MAS domain. MaSE also supplies a method (process) for developing MAS that consists of two major phases: analysis and design.

The purpose of the analysis phase is to provide a set of roles whose tasks meet the system's requirements, i.e., specifying *what* the system should do. The analysis phase consists of the following stages:

- **Capturing goals**, in which the system goals are elaborated and specified from the system point of view in a hierarchical manner.
- **Applying use cases**, in which the system's use cases are specified along with their elaborating sequence diagrams.
- **Refining roles**, in which system functional decomposition is performed by producing a set of roles and their associated tasks. This stage consists of two sub-stages: building the role diagram and specifying the tasks' behavior.

The purpose of the design phase is to specify the way according to which the system-to-be should behave and be constructed. This means, specifying *how* the system will achieve its goals. The design phase consists of the following stages:

- **Creating agent classes**, in which the overall multi-agent system architecture in terms of agents and the conversations among them is determined
- **Constructing conversations**, in which the designer defines the coordination protocols (i.e., conversations) between agent pairs. In particular, two communication class diagrams are defined for each conversation. One diagram specifies the initiator's behavior during that conversation and the other specifies the responder's behavior during that conversation
- **Assembling agents**, in which the internal architecture of the agents is specified
- **System design**, in which the physical system architecture and the distribution of agent classes' instances within that architecture is specified

MaSE is also supported by a CASE tool and was applied in various contexts [55]. Recently, MaSE has shifted into the area of method engineering in which the development process is further customized according to specific needs [54, 56] (see Chap. 9).

MaSE supports all multi-agent concepts except for the mental notion (i.e., BDI). With respect to the modeling and notation aspect, we found out that MaSE supports

well all of the criteria. MaSE also covers most of the development lifecycle. From a pragmatic point of view, MaSE is also equipped with a CASE tool.

## *3.4 PASSI*

PASSI (a Process for Agent Societies Specification and Implementation) is a step-by-step requirement-to-code methodology for designing and developing multi-agent societies, integrating design models and concepts from software engineering approaches and using the UML notation [18].

PASSI consists of five models that should be built sequentially in an iterative manner. In the following, we elaborate on these models and the activities that should be done in order to achieve their purposes.

- The system requirements model aims at describing the system requirements in terms of agents and their goals. It consists of the following:
  - **Domain description**, in which the system functionalities are described using the use case technique
  - **Agent identification**, in which separation of concerns is done by identifying agents using the UML stereotype mechanism. Each agent functionality is represented as a package of use cases
  - **Role identification**, in which each agent is specified by the roles it can play. This is done using class diagram accompanied by the object constraint language
  - **Task specification**, in which the agent behavior is described using the activity diagram of UML
- The agent society model aims at depicting agent interactions and dependencies. It is achieved by performing the following tasks:
  - **Role identification**, in which an elaboration of the previous outcomes is performed by analyzing agent interactions to gather additional understanding on the agent roles
  - **Ontology description**, in which the knowledge that is used within the system is modeled using class diagrams and the stereotype mechanism. In addition, a communication ontology description is achieved within that activity and its purpose is to identify the protocol, the content language, and the ontology mapping
  - **Role description**, in which the roles are described in the context of the agents that play them
  - **Protocol description**, in which protocols that were not defined by FIPA need to be specified using sequence diagrams
- The agent implementation model captures the solution architecture in terms of classes and methods. It is achieved by:

  - **Agent structure definition**, in which one class diagram represents the MAS as a whole, whereas attribute compartments can be used to represent the knowledge of the agent and operations' compartments are used to signify the agent's tasks. In addition, for each agent one class diagram is used to specify the agent's internal structure
  - **Agent behavior description**, in which one or more activity diagrams are drawn to show the flow of events between and within both the main agents' classes and their inner classes (representing their tasks). In addition, the internal behavior of an agent or a task can be specified using flowcharts, activity diagrams, or statecharts

- The code model aims at describing the solution at the code level and consists of the following activities:
  - **Code reuse library**, in which special design patterns are gathered and applied
  - **Code completion baseline,** in which the designer/programmer completes the generate code

- The deployment model aims at specifying the distribution of the parts of the system across hardware processing units, and their migration between processing units. It is done via a deployment configuration diagram.

PASSI is supported by a CASE tool and continuously evolved over the years. It made its progress through patterns [57], agility [58], and tools [59].

PASSI supports all multi-agent concepts except for the mental notion (i.e., BDI). With respect to the modeling and notation aspect, we found that PASSI—as it uses the common modeling language UML—supports well accessibility, expressiveness, and complexity management; however, little information is provided regarding its support for analysis. PASSI also covers most of the development lifecycle. From a pragmatic point of view, PASSI is equipped with a CASE tool and has set plans for continuous improvements.

## *3.5 Prometheus*

The Prometheus methodology aims to provide a means for designing multi-agent systems [20]. It is specifically aimed at building intelligent agents. The Prometheus methodology consists of three phases: (1) the system specification phase that is focused on identifying the basic functionalities of the system, its inputs, its outputs, and its shared data sources; (2) the architectural design phase that is focused on determining the system agents and their interactions; and (3) the detailed design phase that is focused on the internals of each agent. In the following, we elaborate on the activities required in each of the Prometheus phases:

- During the **system specification** phase, the designer is expected to identify the system goals and assign them to roles. She should also spot the actions to be taken

and the precepts to be considered by the system and assign them to roles as well. Finally, she should specify the system functionality using the use case technique; the latter provides a set of scenarios to be associated with the identified roles.

- During the **architectural design** phase, the designer should identify the agent types within the system. In addition, an analysis of the knowledge and data coupling is done along with the agent acquaintance model. Moreover, the agents' lifecycle and description are determined. Referring to the system behavior (phase 1), the interaction protocols are identified in this phase. Finally, the system overview is determined in terms of agents, protocols, events, actions, and shared data.
- During the **detailed design** phase, the internals of the agents should be specified. This should be done by identifying capabilities, plans, internal events, and data. The capabilities can be defined in a hierarchal manner.

The Prometheus methodology also provides a CASE tool that makes it easy to follow its guidelines [60]. The methodology has been integrated into a MAS platform called JACK [61].

Prometheus supports all multi-agent concepts. It supports the modeling and notation to some extent as it introduces additional concepts into the development process. Prometheus also covers most of the development lifecycle. From a pragmatic point of view, Prometheus is equipped with a CASE tool, is taught in courses, is continuously refined [62] and examines additional software engineering aspects to be supported. In Chap. 8, additional research directions of Prometheus are elaborated.

## *3.6 Tropos*

Tropos is an agent-oriented software development methodology founded on two key features: (1) the notion of agent and the associated mentalistic notions (e.g., goals and tasks), and (2) requirements analysis and specification of the system to-be is analyzed with respect to its intended environment [38, 63]. Tropos consists of five development stages:

1. **Early requirements analysis** focuses on the intentions of stakeholders. These intentions are modeled as goals that, through some form of a goal-oriented analysis, eventually lead to the functional and nonfunctional requirements of the system-to-be. The modeling is performed using the i* modeling language, in which stakeholders are represented as (social) actors who depend on each other for goals to be achieved, tasks to be performed, and resources to be furnished.
2. **Late requirements analysis** results in a requirements specification that describes all functional and non-functional requirements for the system-to-be. In Tropos, the system is represented as one or more actors specified in the early requirement stage.

3. **Architectural design** describes how system components work together. Tropos defines organizational architectural styles for cooperative, dynamic, and distributed applications like multi-agent systems, to guide the design of the system architecture. These styles are used to express assertions on the system organizational structure and help match the MAS architecture to the organizational context in which the system will operate.
4. **Detailed design** introduces additional details for each architectural component of a system. In Tropos, one can define how the goals assigned to each actor are fulfilled by agents with respect to pre-defined design patterns.
5. **Implementation** refers to the actual coding of the system-to-be.

Note that Tropos supports the transformational approach in which the transitions from one stage to the next are done by following certain transformational guidelines.

Tropos supports all multi-agent concepts, yet integration with the mental notions (i.e., BDI) requires further examination. It supports the modeling and notation to some extent as it introduces several concepts into the development process that need to be examined. Also, the transitions among the stages require further attention. Tropos also covers most of the development lifecycle. From a pragmatic point of view, Tropos is equipped with a set of tools that provide a suite for specifying, analyzing, and implementing MAS applications. Also, Tropos is continuously evaluated using various techniques.

## *3.7 ADEM*

The agent-oriented development methodology (ADEM) aims at supporting the development of agent-based systems [64]. In particular, ADEM focuses on modeling aspects of agent-based systems using the Agent-Modeling Language (AML) [7, 65]. Similarly to other newer methodologies, ADEM consists of method fragments, techniques, artifacts, and guidelines for creating MAS models. As it is based on RUP, ADEM mainly addresses the business modeling, requirements, and analysis and design workflows. In the following, we list the method fragments provided by ADEM.

| Workflow | Activity |
|---|---|
| Business modeling | Define the business domain ontology |
| | Model business goals |
| | Detail a business actor |
| | Identify business use case responsibility |
| | Structure the extended identify business use case model |
| | Model business organization structure |
| | Model business interactions |
| | Model business services |

(continued)

| Workflow | Activity |
|---|---|
| | Model business observations and affecting interactions |
| | Model business deployment |
| | Detail business architecture |
| | Define the business domain ontology |
| Requirements | Define the domain ontology |
| | Model system goal-based requirements |
| | Detail an actor |
| | Identify use case responsibilities |
| | Structure the extended use case model |
| Analysis and design | Model society |
| | Model interactions |
| | Model interaction ontology |
| | Model services |
| | Model observations and effecting interactions |
| | Detail an entity |
| | Model mental attitudes |
| | Structure behavior |
| | Model deployment |
| | Detail design |

ADEM follows the situational method engineering approach, according to which organizations may adapt their development processes to better fit in. ADEM was developed based on industrial needs within Whitestein Technology, which might position it as better suited for practitioners. However, no evidence is provided of applying the methodology elsewhere. As AML is based on the UML profile, it has several implementations and supporting tools.

It seems that ADEM addresses all multi-agent concepts. It supports the modeling and notation to some extent as it requires the integration of multiple elements and diagram types. Also, the model verification/validation/checking requires more exploration. ADEM also covers most of the development lifecycle. From a pragmatic point of view, ADEM is supported by tools and has been used in various industrial projects.

## 4 Alternative AOSE Methodologies

With the proliferation of AOSE methodologies, the need to integrate these has emerged. In [66] the authors analyzed the meta-models of six methods for developing agent-based systems and found that there is a wide agreement on the concepts of agent-based systems. Following that finding, the authors proposed a unified (and abstract) metamodel that captures the agent-based notions of the six methods.

The resulting metamodeling has many similarities with the metamodels of AML [64] and OPM/MAS [67] in which various aspects of agent-based systems are captured. Another similar effort results in FAML—a generic metamodel for MAS development [68]. Following [69], in which the authors call for standardization of AOSE methodology concepts that will serve as the basis for the next generation methodology, similar to what has been achieved with UML, FAML is a potential candidate for such an effort.

Nevertheless, there in a common understanding that one solution cannot fit well for all cases. Thus, an alternative approach to handle the diversity in AOSE methodologies has emerged, following the method engineering (ME) notions [70]. The approach refers to each of the AOSE methodologies as a set of method fragments. For setting a specific method, one should select the relevant fragments and glue these together. Note that some of the aforementioned methodologies have already adopted that direction. These include INGENIAS, O-MaSE, and ADEM. Akbari [71] also calls for the adoption of the ME approach. However, the ME-based approaches may result in problems in integrating and gluing fragments into a single coherent method. Thus, further examination of this approach is required.

## 5 Concluding Remarks

There are more than two dozen agent-oriented methodologies. Although differences exist among them, there are many similarities as well. As a result of diversity with similarities, the selection of a methodology for developing agent-based systems and applications is nontrivial. This problem intensifies when industrial development is sought, where specific requirements and constraints apply. Additionally, in many cases insufficient resources are available for issues other than modeling, notation, and development process. In particular, as suggested in [72], much work is needed to allow the quantitative evaluation of the agent-based paradigm and the associated methodologies. Such quantitative evaluation should facilitate better assessment of the advantages of agent-based methodologies over existing paradigms in software analysis, design, and maintenance. Additionally, it seems that existing agent-oriented methodologies focus on the development of new systems and not on other stages and aspects within the system lifecycle. Even with reference to development process, not all phases are well supported (e.g., the testing phase is hardly supported). An example of a non-supported aspect is system maintenance, which is hardly dealt with in agent-oriented methodologies. Yet, another void is the lack of support for a paradigm shift to agent orientation. We believe that the latter is a grand challenge for agent-oriented methodologies. That is, it is necessary—yet challenging—to justify the paradigm shift from existing paradigms such as object-oriented, service-oriented, and business-process-oriented to the agent-oriented paradigm.

## References


1. Graham I, Hederson-Sellers B, Younessi H (1997) The OPEN process specification. Addison-Wesley
2. Kinny D, Georgeff M (1996) Modelling and design of multi-agent systems. In: Proceedings of the third international workshop on agent theories, architectures, and languages (ATAL). Lecture notes in computer science 1193. Springer, pp 1–20
3. Kinny D, Georgeff M, Rao A (1996) A methodology and modelling technique for systems of BDI agents. In: Proceedings of the seventh European workshop on modelling autonomous agents in a multi-agent world. Lecture notes in computer science 1038. Springer, pp 56–71
4. Iglesias CA, Garrijo M, Gonzalez J, Velasco JR (1998) Analysis and design of multiagent systems using MAS-CommonKADS. In: Proceedings of the fourth international workshop on agent Theories, architectures and languages (ATAL). Lecture notes in computer science 1365. Springer, pp 313–328
5. Bernon C, Gleizes MP, Picard G, Glize P (2002) The Adelfe methodology for an intranet system design. In: Proceedings of the fourth international bi-conference workshop on agent-oriented information systems (AOIS)
6. Lind J (2001) Iterative software engineering for multiagent systems - The MASSIVE method. In: Lecture notes in computer science 1994. Springer
7. Trencanský I, Cervenka R (2005) Agent modeling language (AML): a comprehensive approach to modeling MAS. Informatica (Slovenia) 29(4):391–400
8. Caire G, Leal F, Chainho P, Evans R, Garijo F, Gomez J, Pavon J, Kearney P, Stark J, Massonet P (2002) Agent oriented analysis using MESSAGE/UML. In: Proceeding of the second international workshop on agent-oriented software engineering May 2001. Lecture notes in computer science 2222. Springer, pp 119–135
9. Jennings NR, Faratin P, Johnson MJ, O'Brien P, Wiegand ME (1996) Using intelligent agents to manage business processes. In: Proceedings of first international conference and exhibition on the practical application of intelligent agents and multiagents, pp 345–360
10. Jennings NR, Faratin P, Norman TJ, O'Brien P, Odgers B (2000) Autonomous agents for business process management. Int J Appl AI 14(2):145–189
11. Jennings NR, Faratin P, Norman TJ, O'Brien P, Odgers B, Alty JL (2000) Implementing a business process management system using ADEPT: a real-world case study. Int J of Appl AI 14(5):421–465
12. Huget M-P (2002) Nemo: an agent-oriented software engineering methodology. In: Proceedings of the OOPSLA 2002 workshop on agent-oriented methodologies
13. Burmeister B (1996) Models and methodology for agent-oriented analysis and design. In: Fischer K (ed) KI'96 Workshop on agent-oriented programming and distributed artificial intelligence, DFKI document D-96-06, http://www.dfki.uni-kl.de/dfkidok/publications/D/96/06/abstract.html
14. Gervais M-P (2003) ODAC: an agent-oriented methodology based on ODP. J Autonom Agent Multi-Agent Syst 7(3); 199–228
15. Wagner G (2003) The agent-object-relationship metamodel: towards a unified view of state and behaviour. Inform Syst 28(5):475–504
16. Debenham J, Henderson-Sellers B (2002) Full lifecycle methodologies for agent-oriented systems - the extended open process framework. In: Proceedings of the fourth international bi-conference workshop on agent-oriented information systems (AOIS)
17. Collinot A, Drogoul A (1998) Using the Cassiopeia method to design a Robot Soccer Team. Appl Artif Intell 12(2–3):127–147
18. Cossentino M (2005) From requirements to CODE with the PASSI methodology. In: Henderson-Sellers B, Giorgini P (eds) Agent-oriented methodologies. Idea Group Inc., Hershey, PA, USA
19. Glaser N (1996) Contribution to knowledge modelling in a multi-agent framework -the CoMoMAS approach- PhD Thesis, L'Universite Henri Poincare

20. Padgham L, Winikoff M (2005) Prometheus: a practical agent-oriented methodology. In: Henderson-Sellers B, Giorgini P (eds) Agent-oriented methodologies. Idea Group Inc., Hershey, PA
21. Brazier FMT, Dunin-Keplicz B, Jennings NR, Treur J (1997) DESIRE: modelling multi-agent systems in a compositional formal framework. Int J Cooperat Inform Syst 6:67–94
22. Brazier FMT, Dunin-Keplicz B, Treur J, Verbrugge LC (1999) Modeling internal dynamic behaviour of BDI agents. In: Meyer JJCh, Schobbes PY (eds) Formal models of agents. Lecture notes in computer science 1760. Springer, pp 36–56
23. Brazier FMT, Jonker CM, Treur J, Wijngaards NJE (1998) Compositional design of a generic design agent. In: Luger G, Interrante L (eds) Proceedings of AAAI workshop on ai and manufacturing: state of the art and state of practice. AAAI Press, pp 30–39
24. Juan T, Pearce A, Sterling L (2002) ROADMAP: extending the GAIA methodology for complex OPEN systems. In: Proceedings of AAMAS '02. pp 3–10
25. d'Inverno M, Kinny D, Luck M, Wooldridge M (1997) A formal specification of dMARS. In: Proceedings of the fourth international workshop on agent theories, architectures and languages (ATAL). Lecture notes in computer science 1365. Springer, pp 155–176
26. d'Inverno M, Luck M (1997) Development and application of a formal agent framework. In: Proceedings of the first IEEE international conference on formal engineering methods. pp 222–231
27. d'Inverno M, Luck M (2004), Understanding agent systems. Springer
28. Luck M, d'Inverno M (1995) Structuring a Z specification to provide a formal framework for autonomous agent systems. In: Proceedings. of ZUM '95. Lecture notes in computer science 967. Springer, pp 47–62
29. Luck M, Griffiths N, d'Inverno M (1996) From agent theory to agent construction: a case study. In: Proceedings of third international workshop on agent theories, architectures and languages (ATAL). Lecture notes in computer science 1193. Springer, pp 49–63
30. Sierra C, Sabater J, Agustí J, Garcia P (2002) Evolutionary programming in SADDE. In: Proceedings of the first international joint conference on autonomous agents and multi agent systems (AAMAS). pp 1270–1271
31. Wooldridge M, Jennings NR, Kinny D (2000) The Gaia methodology for agent-oriented analysis and design. J Autonom Agent MAS 3(3):285–312
32. Zambonelli F, Jennings N, Wooldridge M (2001) Organizational rules as an abstraction for the analysis and design of multiagent systems. Int J Software Eng Knowledge Eng 11(4):303–328
33. Zambonelli F, Jennings NR, Wooldridge M (2003) Developing multiagent systems: the Gaia methodology. ACM Trans on Software Eng Methodol 12(3):317–370
34. Omicini A (2001) SODA: societies and infrastructures in the analysis and design of agent-based systems. In: Proceedings of the first international workshop on agent-oriented software engineering (AOSE). Lecture notes in computer science 1957. Springer, pp 185–194
35. Pavón JJ, Gómez-Sanz JJ, Fuentes R (2005) The INGENIAS methodology and tools. In: Henderson-Sellers B, Giorgini P (eds) Agent-oriented methodologies. Idea Group Inc., Hershey, PA
36. Bush G, Cranefield S, Purvis M (2001) The Styx agent methodology, The Information Science Discussion Paper Series 2001/02. Department of Information Science, University of Otago, New Zealand
37. Abdelaziz T, Elammari M, Unland R, Branki C (2010) MASD: multi-agent systems development methodology. Multiagent Grid Syst J 6(1):71–101
38. Bresciani P, Giorgini P, Giunchiglia F, Mylopoulos J, Perini A (2004) TROPOS: an agent-oriented software development methodology. J Autonom Agent Multi-Agent Syst 8(3):203–236
39. DeLoach SA, Wood MF, Sparkman CH (2001) Multiagent systems engineering. Int J Software Eng Knowledge Eng 11(3):231–258
40. Yu E, Cysneiros M (2002) Agent-oriented methodologies—towards a challenge Exemplar. In: Proceedings of the 4th Intl. Workshop on agent-oriented information systems (AOIS'02)

41. Cernuzzi L, Rossi G (2002) On the evaluation of agent oriented methodologies. In: Proceedings of the OOPSLA 2002 workshop on agent-oriented methodologies
42. Shehory O, Sturm A (2001) Evaluation of modeling techniques for agent-based systems. Agents 2001:624–631
43. Dam HK, Winikoff M (2004) Comparing agent-oriented methodologies, AOIS 2003. Lect Notes Comput Sci 3030:78–93
44. Sturm A, Shehory O (2003) A framework for evaluating agent-oriented methodologies, AOIS 2003. Lecture notes in computer science 3030. pp 94–109
45. Cuesta P, Gómez A, González JC, Rodríguez FJ (2003) a framework for evaluation of agent oriented methodologies. In: The conference of the Spanish Association for Artificial Intelligence (CAEPIA)
46. Garcia E, Giret A, Botti V (2011) Evaluating software engineering techniques for developing complex systems with multiagent approaches. Inform Software Technol 53(5):494–506
47. Tran QN, Low G (2005) Comparison of ten agent-oriented methodologies. In: Henderson-Sellers B, Giorgini P (eds) Agent-oriented methodologies, vol 12, Idea Group Publishing., pp 341–367
48. Cernuzzi L, Zambonelli F (2011) Improving comparative analysis for the evaluation of AOSE methodologies. IJAOSE 4(4):331–352
49. Cernuzzi L, Molesini A, Omicini A, Zambonelli F (2011) Adaptable multi-agent systems: the case of the Gaia methodology. Int J Software Eng Knowledge Eng 21(4):491–521
50. Cernuzzi L, Zambonelli F (2009) Gaia4E: a tool supporting the design of MAS using Gaia. ICEIS 4:82–88
51. García-Ojeda J, Arenas A, Pérez-Alcázar J (2005) Paving the way for implementing multiagent systems: refining GAIA with AUML. In: Proceedings of the 6th international workshop (AOSE2005). Lecture notes in computer science 3950. Springer, pp 179–189
52. Moraitis P, Spanoudakis N (2006) The GAIA2JADE process for multi-agent systems development. Appl Artif Intell 20(2–4):251–273
53. Spanoudakis N, Moraitis P (2009) Gaia agents implementation through models transformation. In: Proceedings of the 12th international conference on principles of practice in multi-agent systems (PRIMA '09). Springer, pp 127–142
54. DeLoach SA, García-Ojeda JC (2010) O-MaSE: a customisable approach to designing and building complex, adaptive multi-agent systems. IJAOSE 4(3):244–280
55. DeLoach SA, Wood M (2001) Developing multiagent systems with agentTool. In: Proceedings of the seventh international workshop on agent theories, architectures, and languages (ATAL). Lecture notes in computer science 1986. Springer, pp 46–60
56. Juan C. García-Ojeda, DeLoach SA, Robby: agentTool process editor: supporting the design of tailored agent-based processes. In: Proceedings of SAC 2009. pp 707–714
57. Cossentino M, Sabatucci L, Sorace S, Chella A (2003) Patterns reuse in the PASSI methodology. In: Fourth international workshop engineering societies in the agents World (ESAW '03)
58. Chella A, Cossentino M, Sabatucci L, Seidita V (2006) Agile PASSI: an agile process for designing agents. Int J Comput Syst Sci Eng. Special issue on "Software Engineering for Multi-Agent Systems" 21(2)
59. Chella A, Cossentino M, Sabatucci L (2004) Tools and patterns in designing multi-agent systems with PASSI. WSEAS Trans Commun 3(1):352–358
60. Padgham L, Thangarajah J, Winikoff M (2007) The prometheus design tool - a conference management system case study DOI:10.1007/978-3-540-79488-2_15. In: Agent-oriented software engineering VIII DOI:10.1007/978-3-540-79488-2: 8th International Workshop, AOSE 2007. Lecture notes in computer science 4951. Springer, pp 197–211
61. Winikoff M (2005) JACK intelligent agents: an industrial strength platform. In: Multi-agent programming: languages, platforms, and applications. Springer, pp 175–193
62. Khallouf J, Winikoff M (2009) Goal-oriented design of agent systems: a refinement of prometheus and its evaluation. Int J Agent-Oriented Software Eng 3(1):88–112
63. Castro J, Kolp M, Mylopoulos J (2002) Towards requirements-driven information systems engineering: the Tropos Project. Inform Syst 27(6):365–389

64. Cervenka R, Trencansky I (2007). The agent modeling language - AML: a comprehensive approach to modeling multi-agent systems (Whitestein Series in Software Agent Technologies and Autonomic Computing). Birkhäuser
65. Cervenka R (2012) Modeling multi-agent systems with AML. Software Agents, Agent Systems and Their Applications 2012, NATO, pp 9–27
66. Bernon C, Cossentino M, Pavon J (2006) Agent-oriented software engineering. Knowledge Eng Rev 20(2):99–116
67. Sturm A, Dori D, Shehory O, An object-process- based modeling language for multiagent systems. IEEE Trans Syst Man and Cybern—Part C: Appl Rev 40(2);227–24
68. Beydoun G, Low G, Henderson-Sellers B, Mouratidis H, Gomez-Sanz J-J, Pavon J, Gonzalez-Perez C (2009) FAML: a generic metamodel for MAS development. IEEE Trans Software Eng 35(6):841–863
69. Dam HK, Winikoff M (2013) Towards a next-generation AOSE methodology. Sci Comput Program 78(8):684–694
70. Henderson-Sellers B, Ralyte J (2010) Situational method engineering: state-of-the-art review. J Universal Comput Sci 16(3):424–478
71. Akbari OZ (2010) A Survey of agent-oriented software engineering paradigm: towards its industrial acceptance. J Comput Eng Res 1(2):14–28
72. Zambonelli F, Omicini A (2004) Challenges and research directions in agent-oriented software engineering. J Autonom Agent Multi-Agent Syst 9(3):253–287