O-MaSE: a customisable approach to designing and building complex, adaptive multi-agent systems

Scott A. DeLoach*

Kansas State University,
234 Nichols Hall, Manhattan,
Kansas, KS 66506, USA
E-mail: sdeloach@k-state.edu
*Corresponding author

Juan Carlos García-Ojeda

School of Systems Engineering,
School of Natural Sciences and Engineering,
Autonomous University of Bucaramanga,
Street 48 #39-234, Bucaramanga, Colombia
E-mail: jgarciao@unab.edu.co

Abstract: The complexity and scope of software systems continues to grow. One approach to dealing with this growing complexity is the use of intelligent, multi-agent systems. However, due in part to its relative infancy when compared to other software paradigms, the use of multi-agent systems has yet to be used extensively in industry. One reason is the lack of industrial strength methods and tools to support multi-agent development. This paper presents the organisation-based multi-agent software engineering (O-MaSE) methodology framework, which integrates a set of concrete technologies aimed at facilitating industrial acceptance. Specifically, O-MaSE is a customisable agent-oriented methodology based on consistent, well-defined concepts supported by plug-ins to an industrial strength development environment, agentTool III.

Keywords: agent-oriented methodology; method engineering; integrated development environments; software analysis; software design.

Reference to this paper should be made as follows: DeLoach, S.A. and García-Ojeda, J.C. (2010) 'O-MaSE: a customisable approach to designing and building complex, adaptive multi-agent systems', *Int. J. Agent-Oriented Software Engineering*, Vol. 4, No. 3, pp.244–280.

Biographical notes: Scott DeLoach is an Associate Professor in the Computing and Information Sciences Department at Kansas State University, USA. His research focuses on methods and techniques for the analysis, design and implementation of complex adaptive systems, which have been applied to both multi-agent and cooperative robotic systems. He is best known for his work in agent-oriented software engineering. He is the creator of the multi-agent systems engineering methodology (MaSE), its follow-on organisation-based multi-agent systems engineering methodology (O-MaSE), and the associated agentTool analysis and design tool. He has more than 75 refereed publications and has advised over 25 graduate students. He came to Kansas State University after a 20-year career in the US Air Force.

Juan C. García-Ojeda is an Associate Professor in the Systems Engineering Department at the Universidad Autónoma de Bucaramanga (UNAB), Colombia. His research interests include agent-oriented software engineering, web programming, method engineering, meta-modelling, and agent synthesis. In 2006, he was awarded with a Fulbright Scholarship to pursue graduate studies in the USA. From 2006 to 2009, he worked under the supervision of Dr. Scott A. DeLoach in the definition of a framework for constructing customisable agent-oriented software development processes. In 2010, he has been included in the *27th Edition of Who´s Who in the World* for his outstanding achievements in his field of endeavour and contributed to the progress of modern society.

1 Introduction

Today's software industry is tasked with building ever more complex software applications. While software development methods and techniques have made great strides over the last 30 years, the demand being placed on software is increasing even more rapidly. Businesses today are demanding applications that operate autonomously, adapt in response to dynamic environments, and interact with other distributed applications in order to provide wide-ranging solutions (Jennings et al., 1998; Luck et al., 2005). This insatiable demand has left the software industry constantly looking for new computing metaphors and approaches to allow it to cope.

Multi-agent system (MAS) technology is a promising approach capable of meeting these new demands (Luck et al., 2005). Its central notion – the intelligent agent – encapsulates the appropriate characteristics (i.e., autonomy, pro-activity, reactivity, and interactivity) necessary to meet the requirements of these new applications. Unfortunately, there is a disconnection between the advanced technology being created by the multi-agent community and its application in industrial software. The obstacles to industrial adoption have been the focus of several discussions. Jennings et al. (1998) mention two major obstacles to widespread adoption of agent technologies in industry:

1 the lack of complete methodologies and processes to help designers to specify, analyse, and design agent-based applications

2 the lack of industrial-strength agent-based toolkits.

Luck et al. (2005) also suggest that the lack of mature methodologies and programming tools are the culprit. In a special session at AAMAS 2008, leading MAS researchers and engineers were asked to discuss the obstacles currently impeding industrial adoption of MAS technology. While there were a variety of opinions, Georgeff (2009) and DeLoach (2009a) suggested that standard definitions of agent concepts and agent-oriented methodologies are one of the keys to advancing MAS into the mainstream while Winikoff (2009) and Calisti and Rimassa (2009) both argued for producing concrete tools to support MAS techniques and methodologies.

Odell et al. (2001) advise that acceptance of any new technology requires techniques to reduce the inherent risk of that technology. They go on to assert that acceptance of new software development methods requires standard representations for artefacts supporting

analysis, specification, and design. Thus, they propose two approaches for gaining industry acceptance of MAS technology. First, they suggest presenting the new methods as incremental extensions to known and trusted methods. Second, they recommend providing engineering tools to support the new methods that are similar to existing industrial practice.

An alternative approach to defining industrial strength methodologies that has gained support in the agent-oriented software engineering community is situational method engineering, which promotes flexibility in MAS methods and processes (Low et al., 2009; Molesini et al., 2009; Cossentino et al., 2007). Henderson-Sellers (2005) was one of the first to argue that situational method engineering was the key to creating industrial strength methodologies as it allows the creation of standard approaches that are widely supported while continuing to allow innovation and research. Situational method engineering allows method engineers to construct methods (a.k.a. methodologies) from a set of existing method fragments (Brinkkemper, 1996).

As method engineering is a young field, several terms are used ambiguously in the literature. Chief among these are method, methodology, process model and process. In this paper, the terms *method* and *methodology* are used synonymously with *process model* while the term *process* is used to denote an instance of a process model or method that is enacted to develop a software system. Some exceptions to this convention exist in the naming of tool components as they have retained their historical names (e.g., the 'agentTool process editor').

This paper presents an overview of the organisation-based multi-agent software engineering (O-MaSE) methodology framework, which integrates a set of concrete technologies aimed at facilitating industrial acceptance through situational method engineering. Specifically, O-MaSE is a customisable agent-oriented methodology based on consistent, well-defined concepts supported by plug-ins to an industrial strength development environment.

The goal of the O-MaSE methodology framework is to allow method engineers to build custom agent-oriented methods using a set of method fragments, all of which are based on a common meta-model. To achieve this, O-MaSE is defined in terms of a meta-model, a set of method fragments, and a set of method construction guidelines. The O-MaSE *meta-model* defines a set of analysis, design, and implementation concepts and a set of constraints between them. The *method fragments* define a set of work products, a set of activities that produce work products, and the performers of those activities. Finally, *method construction guidelines* define how the method fragments may be combined to create O-MaSE compliant methods. In general, an *O-MaSE compliant method* is an instance of the O-MaSE methodology in which appropriate method fragments are assembled into a method such that the method construction guidelines are satisfied. Critical to the O-MaSE methodology framework is the agentTool III (aT^3) integrated development environment that supports the creation of custom O-MaSE compliant methods as well as providing the editors, verification tools, and code generators for creating complex, adaptive systems using MAS technology.

O-MaSE has its roots in the original multi-agent systems engineering (MaSE) methodology (DeLoach et al., 2001). While MaSE provided a good starting point for developing MASs, it had several problems. First, MaSE produced MASs with a fixed organisation. Agents developed in MaSE played a limited number of roles and had a limited ability to change those roles, regardless of their individual capabilities. In addition, MaSE did not include the notion of sub-teams and had no mechanism for

modelling interactions with the environment. Finally, MaSE was utterly inflexible. MaSE prescribed a strict set of models that built upon each other; there were no guidelines to help a developer deviate from the established method. The aT^3 toolset is the successor to the original agentTool that was developed in 2000–2001 to support MaSE (DeLoach and Wood, 2001). The aT^3 toolset is a plug-in to the eclipse platform and extends the eclipse process framework (EPF) to handle method customisation.

While many pressing issues have been tackled in O-MaSE, at least for the moment, many tasks critical for a complete software methodology such as management, product deployment, and testing and evaluation have been intentionally ignored. Management and deployment issues are generally applicable over a wide variety of software projects and thus existing approaches can and should be applied. Testing and evaluation is not yet included in O-MaSE, as current work has focused strictly on the analysis, design, and implementation of MASs; while many traditional techniques can be applied to MASs, the need for unique approaches and tools is recognised. Existing research can be used to extend O-MaSE in this area (Poutakidis et al., 2009; Nguyen et al., 2008; Lam and Barber, 2005; Coelho et al., 2006).

Following a discussion of background material in Section 2, O-MaSE is introduced in Section 3 in terms of its meta-model, method fragments, and guidelines. The aT^3 toolkit is introduced in Section 4 while Section 5 illustrates the use of O-MaSE on two examples. Section 6 presents a comparison of O-MaSE with related methodologies, while Section 7 provides a final discussion and describes future work.

2 Background

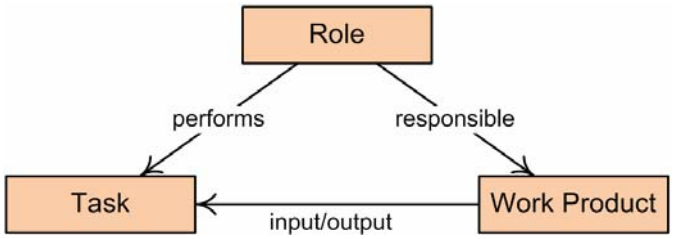
Method engineering is an approach where method engineers construct methods (a.k.a. methodologies) from a set of method fragments as opposed to modifying or tailoring monolithic, 'one-size-fits-all' methods to suit their needs. Method fragments are generally created by extracting useful tasks and techniques from existing methods and redefining them in terms of a common meta-model. The fragments are then stored in a repository for later use. During method creation, method engineers select suitable method fragments from the repository and assemble them into complete methods meeting project specific requirements (Brinkkemper, 1996).

While intuitively straightforward, the application of method engineering for developing agent-oriented applications is non-trivial. Specifically, there is currently no consensus on the main elements distinguishing MASs. While concepts such as agents, roles, and goals appear in many MAS techniques and methodologies, the definitions of those concepts are inconsistent and often unrelated. Thus, Beydoun et al. (2005) (along with others) have suggested that prior to developing a set of method fragments, a well defined meta-model of common agent-oriented concepts should be developed and agreed upon similar to the object-oriented community.

Three similar meta-models exist to help apply method engineering to the production of custom methods: SPEM 2.0 (OMG, 2008), OPEN (Firesmith and Henderson-Sellers, 2002), and SEMDM (a.k.a. ISO/IEC 24744) (ISO/IEC, 2007). The software and systems process engineering meta-model (SPEM) is "a process engineering meta-model as well as conceptual framework, which can provide the necessary concepts for modelling, documenting, presenting, managing, interchanging, and enacting developments processes" (OMG, 2008). SPEM distinguishes between reusable *method content* and the

way it is applied in actual *methodologies*. SPEM method content captures and defines the key tasks, roles, and work products[1] that are used in a software development methodology. As shown in Figure 1, tasks define the work that is performed by roles to use an input set of work products to create and output set of work products.

Figure 1 Key SPEM 2.0 method content concepts (see online version for colours)



Source: Diagram derived from Firesmith and Henderson-Sellers (2002)

Development methodologies are assembled into a set of activities, populated with tasks and their associated roles and work products. Thus, activities are aggregates of either basic content or other activities. SPEM defines three special types of activities: phases, iterations and processes. Phases are special activities that take a period of time and end with a major milestone or set of work products. Iterations are activities that group other activities that are often repeated. Finally, processes are special activities that specify the structure of a software development project.

In a similar vein, the OPEN process framework (OPF) uses a meta-model-based framework that allows designers to select method fragments from a repository in order to construct custom methods (Firesmith and Henderson-Sellers, 2002). The OPF is defined in three layers: M2, M1, and M0. The M2 layer includes the OPF meta-model, which defines the types of method fragments that can be created. The OPF meta-model defines methodologies as consisting of stages, work units (activities, tasks, and techniques), producers, work products, and languages. The M1 layer includes a repository of method fragments and a methodology specific meta-model defining the concepts used within those fragments. The method engineer uses predefined method fragments from M1 to creating custom methods that are enacted at the M0 level on a specific project.

ISO/IEC 24744 defines the software engineering meta-model for development methodologies (SEMDM), a competing meta-model for defining methodologies. SEMDM is unique in its ability to formalise the notion of dual-layer modelling using *powertypes* (Gonzalez-Perez and Henderson-Sellers, 2006). Dual layer modelling refers to the situation where instances of methodology concepts (e.g., requirements specification, architectural design) are used as classes by developers to create instances of those classes (e.g., specific specifications and designs) during the enactment of the methodology. SEMDM defines methodologies as consisting of templates of stages, work units, work products, model units, and producers along with a set of resources, which define the languages, notations, constraints, outcomes and guidelines used. SEMDM also defines an action that captures whether particular task of a work unit creates, modifies, or uses specific work products.

The core concepts of SPEM, OPF and SEMDM are parallel. SPEM roles are essentially OPF and SEMDM producers; SPEM activities are similar, but not identical, to OPF and SEMDM work units; work products are analogous between the three. The

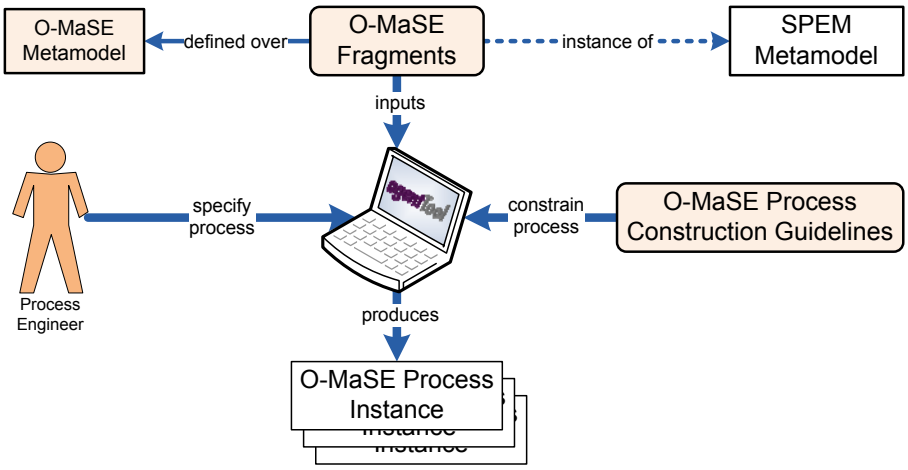
difference between activities and work units is that OPF and SEMDM work units describe *what* is to be done, but not *when* while SPEM mixes the two. O-MaSE was originally defined using the OPF. However, due to the popularity of SPEM in the agent-oriented software engineering community and the use of the SPEM-based EPF to implement the aT^3 process editor (APE) (see Section 4), O-MaSE has been redefined here in terms of SPEM 2.0.

In a related effort, the Foundation for Physical Agents Technical Committee (FIPA-TC) Methodology group attempted to define reusable method fragments from existing agent-oriented methodologies (Seidita et al., 2006). As part of this effort, the group is currently defining a design process documentation template (http://www.fipa.org/subgroups/DPDF-WG.html), which uses SPEM 2.0 as its base.

3 The O-MaSE methodology framework

The O-MaSE methodology framework is based on two meta-models: SPEM 2.0 and the O-MaSE meta-model. The SPEM meta-model defines methodology-related concepts while the O-MaSE meta-model defines the product related concepts. As shown in Figure 2, the definition of O-MaSE consists of three main components: the O-MaSE meta-model, method fragments, and guidelines. In general, a method engineer creates new O-MaSE compliant methods in aT^3 by selecting O-MaSE fragments and combining them into a method that is consistent with the method construction guidelines. O-MaSE fragments are instances of SPEM elements such as tasks, work products, and roles, and are defined in terms of concepts from the O-MaSE meta-model. For example, the O-MaSE role model is an instance the SPEM work product and is defined in terms of roles, goals and capabilities, each of which are defined in the O-MaSE meta-model. In this section, the three O-MaSE components are defined. First, the O-MaSE meta-model is defined. Next, a discussion of O-MaSE phases is given followed by an explanation of the method fragments. Finally, the guidelines governing the construction of O-MaSE compliant methods are examined.

Figure 2 O-MaSE methodology framework (see online version for colours)

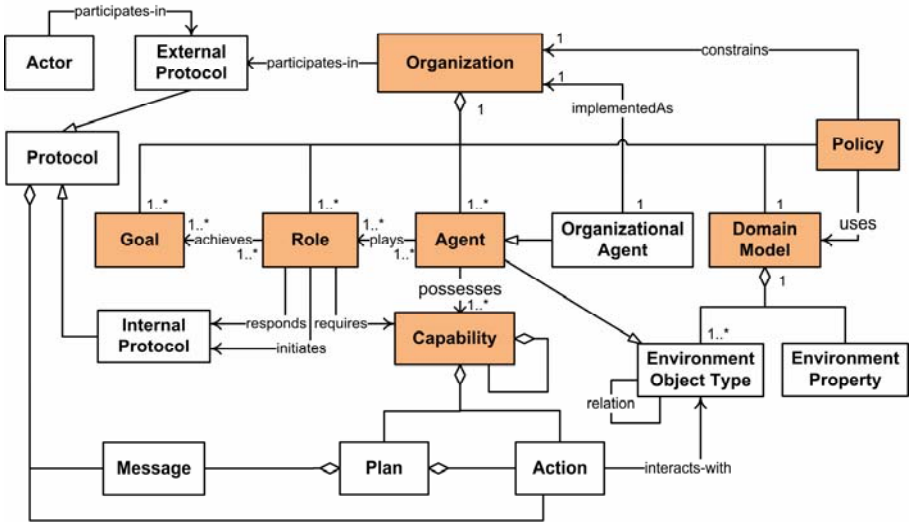


3.1 Meta-model

The O-MaSE meta-model defines the main concepts and relationships used to define MASs. The O-MaSE meta-model is based on an organisational approach (DeLoach and Valenzuela, 2007; DeLoach et al., 2008) and includes notions that allow for hierarchical, holonic, and team-based decomposition of organisations (Horling and Lesser, 2004). The O-MaSE meta-model was derived from the organisation model for adaptive computational systems (OMACS). OMACS captures the knowledge required of a system's organisational structure and capabilities to allow it to organise and reorganise at runtime (DeLoach et al., 2008). The key decision in OMACS-based systems is determining which agent to assign to which role in order to achieve which goal.

Using models such as OMACS at runtime has recently become an important research area as it and allows efficient and effective runtime adaptation (Blair et al., 2009). While O-MaSE does not focus solely on OMACS-based systems, O-MaSE does provide direct support for such systems. As shown in Figure 3, an *organisation* is composed of five entities: goals, roles, agents, domain model, and policies (shaded entities correspond directly to OMACS entities and multiplicities of 0..* are omitted for clarity). Each of these entities is discussed below.

Figure 3 O-MaSE meta-model (see online version for colours)



Note: Shaded entities are from OMACS.

In the traditional artificial intelligence sense, a *goal* represents a desirable state (Russell and Norvig, 2003) or the objective of a computational procedure (van Lamsweerde et al., 1998). In agent-oriented circles, van Riemsdijk et al. (2008, p.714) define a goal as "a mental attitude representing preferred progressions of a particular MAS". This definition captures the notion of individually distinct goals that require specific actions to reach a particular state. As such, O-MaSE uses goals to define the objectives of the organisation. A *role* defines a position within an organisation whose behaviour is expected to *achieve* a particular goal or set of goals (due to the naming conflict between O-MaSE roles and

methodology-related roles, the term *method-role* is used to refer to methodology-related roles throughout the remainder of this paper). Agents are assigned to play those roles and perform the behaviour expected of those roles. *Agents* are autonomous entities that can perceive and act upon their environment (Russell and Norvig, 2002). To carry out perception and action, an agent possesses a set of capabilities. *Capabilities* can be used to capture soft abilities (i.e., algorithms) or hard abilities (i.e., physical sensors or effectors). An agent that *possesses* all the capabilities *required* to play a role, may be assigned that role in the organisation. Capabilities can be defined as

1 a set of sub-capabilities

2 a set of actions that may interact with the environment

3 a plan that uses actions in specific ways.

Organisational agents (OAs) are organisations that act as agents in a higher-level organisation and thus capture the notion of organisational hierarchy. As agents, OAs may possess capabilities, coordinate with other agents, and be assigned to play roles. OAs are similar to the notion of non-atomic holons in the ASPECS methodology (Cossentino et al., 2009). Therefore, OAs represent an extension to the traditional agent-group-role (AGR) model (Ferber and Gutknecht, 1998; Ferber et al., 2003) and the organisational meta-model proposed by Odell et al. (2005).

The *domain model* is used to capture the key elements of the environment in which agents will operate. These elements are captured as *domain object types* from the environment, which includes agents, and the *relationships* between those object types. It can also be used to capture general *environment properties* that describe how the objects behave and interact (DeLoach and Valenzuela, 2007). A designer may use entities defined in the O-MaSE model (goals, roles, agents, etc.) along with entities defined in the domain model to specify organisational *policies* to *constrain* how an organisation may behave in a particular situation. Policies are often used to specify liveness and safety properties of the system being designed.

Protocols define interactions between roles or between the organisation and external *actors*. Protocols are generally defined as patterns of communication between such entities (Odell et al., 2001, 2000). A protocol can be of two types, external or internal. *External protocols* specify interactions between the organisation and external actors (i.e., humans or other software applications), while *internal protocols* specify interactions between agents playing specific roles in the organisation. Either messages or actions can be used to define protocols. Messages are typically used for communications; however, actions may be used to modify the environment as a means of communication (Holland and Melhuish, 1999).

3.2 Phases

SPEM uses phases to organise the various activities of a development method. While O-MaSE explicitly defines activities and tasks (see overview in Table 1), it does not define specific phases. Because there are numerous ways to organise activities, O-MaSE makes no commitments to a predefined set of phases. Instead, O-MaSE allows method engineers to organise Activities in different ways based on project need. For instance, O-MaSE has been used to support modern iterative, incremental approaches as proposed by Royce (1998) and as implemented in the popular rational unified process (RUP) (Kroll

and Kruchten, 2003; Kruchten, 2000). However, O-MaSE has also been used in several projects using much simpler approaches such as the classical waterfall model (Royce, 1970).

Table 1 O-MaSE method fragments

Activities	*Tasks*	*Work products created/modified*	*Responsible method-roles*
Requirements gathering	Requirements specification	Requirements spec	Requirements engineer
Problem analysis	Model goals	Goal model	Goal modeller
	Refine goals		
	Model domain	Domain model	Domain modeller
Solution analysis	Model organisation interfaces	Organisation model	Organisation modeller
	Model roles	Role model	Role modeller
	Define roles	Role description document	
	Define role goals	Role goal model	
Architecture design	Model agent classes	Agent class model	Agent class modeller
	Model protocols	Protocol model	Protocol modeller
	Model policies	Policy model	Policy modeller
Low level design	Model plans	Agent plan model	Plan modeller
	Model capabilities	Capabilities model	Capabilities modeller
	Model actions	Action model	Action modeller
Code generation	Generate code	Source code	Programmer

Figure 4 shows an example of using an iterative, incremental approach with O-MaSE. Here, the goal of the *inception phase* is to establish what is and is not part of the product. The inception phase is broken into two iterations, the first focusing solely on problem analysis, while the second continues to refine the problem analysis while doing some preliminary solution analysis. The *elaboration phase*, whose goal is to demonstrate an architecture that can support key requirements, is also broken into two iterations. In Iteration 3, the solution analysis is further refined while initial architecture design work begins. In Iteration 4, solution analysis is finalised, more architecture design is carried out, and preliminary low level design is done to support an executable prototype. The goal of the *construction phase* is to produce an acceptable version of the system within cost and schedule. It starts with Iteration 5 where the architecture design is finalised and the low level design and code generation of the initial features is performed. Iteration 6 continues with the low level design and code generation for the next set of features.

Figure 5 shows an example of using O-MaSE with a waterfall approach. In this case, there are three main phases: requirements analysis, design, and implementation. In this case, the main activities are allocated as expected, with problem and solution analysis done in the requirements analysis phase, architecture and low level design done during the design phase and code generation done during the implementation phase.

Figure 4 Using iterative, incremental phases in O-MaSE (see online version for colours)

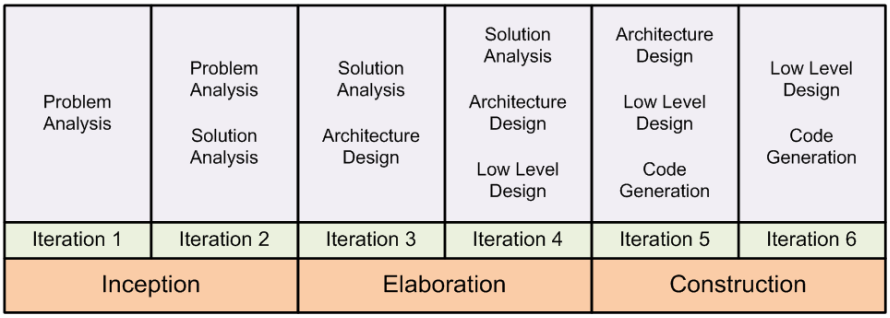
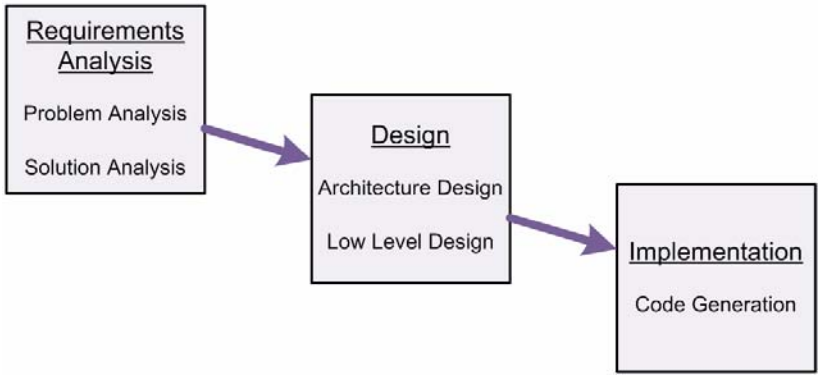


Figure 5 Using waterfall phases with O-MaSE (see online version for colours)



Therefore, the definition of a complete O-MaSE compliant method requires the method engineer to distribute activities and tasks to phases, as defined by the overall approach (iterative, incremental, waterfall, etc.). As this will be unique for each system being developed, there are no hard and fast rules on what activities should be placed in which phase. However, there are dependencies between the various fragments that must be maintained. These dependencies are captured as method construction guidelines as described in Section 3.6. As the construction of O-MaSE compliant methods can be somewhat confusing, method construction is supported by the APE as described in Section 4.2; this support includes automated validation of methods using the method construction guidelines.

3.3 Activities

Table 1 shows six activities currently covered by O-MaSE: requirements gathering, problem analysis, solution analysis, architecture design, low level design, and code generation. *Requirements gathering* is the process of identifying software requirements from a variety of sources. Typically, requirements are classified as either functional requirements, which define the functions required by the software, or non-functional requirements, which specify traits of the software such as performance quality and usability that are not directly related to software function.

The goal of the *problem analysis* is to capture the purpose of the product and document the environment in which it will be deployed. O-MaSE captures this information in a goal model, which captures the purpose of the product, and a domain model that captures the environment in which the product exits. The objective of *solution analysis* is to translate the purpose and environment of the project into a description of the required system behaviour and interactions with external entities such as users and existing systems. This behaviour is captured as roles and interactions in the organisation model and role model.

Once the goals, environment, behaviour, and interactions of the system are known, *architecture design* is used to create a high-level description of the main system components and their interactions. The architecture is captured in agent class models, protocols, and policies. This high-level description is then used to drive *low level design*, where the detailed specification of the internal agent behaviour is defined. Low-level agent behaviour is captured in plan, capability and action models. This low-level specification of agent behaviour is then used to implement the individual agents during *code generation*. While not currently defined in O-MaSE, system creation ends with testing, evaluation, and deployment of the system. Fortunately, the nature of the O-MaSE framework allows it to be extended based on current research and state of practice methods and techniques and thus incorporation of these activities is straightforward.

3.4 Tasks

Next, the typical tasks, work products, and method-roles used in O-MaSE are defined. While Table 1 shows tasks as being associated with specific activities, this is not always the case. As with the allocation of activities to phases, O-MaSE does not require specific Tasks to be performed in specific activities. For instance, even though the model protocols task is generally part of the architecture design activity, there is nothing to preclude a method engineer from including it in a solution analysis activity to define the protocols between roles defined in a role model. The only hard and fast requirements are contained in the method construction guidelines in Section 3.6.

Each task is defined below with a general description of the task objective along with a description of the steps used to produce the associated work products.

Throughout this paper, O-MaSE concepts, tasks, and models are illustrated using a *temperature monitoring system* (TMS) example as derived from (Bakshi et al., 2005). The TMS is a distributed, sensor system, where each node has a processor and a temperature system. During operation, each node monitors the temperature gradient between itself and its neighbours (those within one-hop). If this temperature gradient exceeds a given threshold, a *local alarm* occurs; if the node can corroborate this reading with a larger set of neighbours (those within ten metres) it triggers a *global alarm*. Each node is responsible to ‘push’ its temperature reading to its neighbours at a set rate. However, when a node needs to corroborate a temperature gradient, the node is required to ‘pull’ that data from all nodes within ten metres.

3.4.1 Requirements specification

There are several techniques for gathering software requirements. In general, there are several steps in requirements specification including elicitation, analysis, specification, negotiation, and validation. In many cases, traditional techniques (Pressman, 2010) for

gathering requirements (e.g., data flow diagrams, use cases, and event-response tables) will be sufficient, while in other cases newer approaches focused toward multi-agent systems are applicable (Castro et al., 2002; Fuentes-Fernández et al., 2009). O-MaSE assumes that either traditional or multi-agent focused requirements gathering techniques are sufficient and thus does not stipulate a specific technique; the method engineer is free to use any existing technique deemed appropriate.

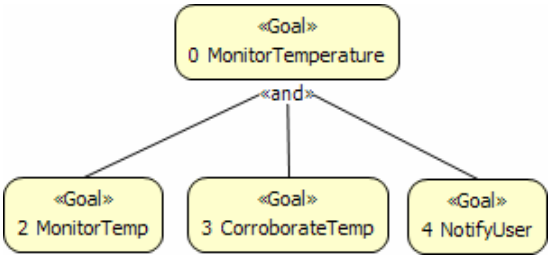
3.4.2 Model goals

The objective of the model goals task is to transform the initial system requirements into a set of structured goals for the system. Goal models are widespread in many agent-oriented methodologies (DeLoach et al., 2001; Giorgini et al., 2005; Padgham and Winikoff, 2002). The deliverable of the model goals task is an initial goal model.

The typical approach to modelling goals is AND/OR decomposition (van Lamsweerde and Letier, 2000). The objective of this approach is to refine the overall goal of the system into a set of sub-goals. If all the sub-goals must be achieved in order to achieve the parent goal, the parent is AND-refined, while if the sub-goals represent alternative ways to achieve the parent goal, the parent goal is OR-refined.

An O-MaSE goal model for the TMS system is shown in Figure 6. The overall goal, monitor temperature is AND-refined into three sub-goals: MonitorTemp, CorroborateTemp, and NotifyUser. Essentially, the goal model creates a high-level specification of *what* the system should do. Each goal in the model is annotated by the keyword 'goal'. A line between two goals with an 'and' keyword at the parent end is used to represent AND-refinement while a line with an 'or' keyword at the parent is used to represent OR-refinement.

Figure 6 Goal model (see online version for colours)



3.4.3 Refine goals

The refine goals task captures the dynamic aspects of the goal model and further defines each goal using a technique called attribute-precede-trigger analysis. The result is a refined version of the goal model called a goal model for dynamic systems (GMoDS) goal model (DeLoach and Miller, 2010).

The refine goals task is used to

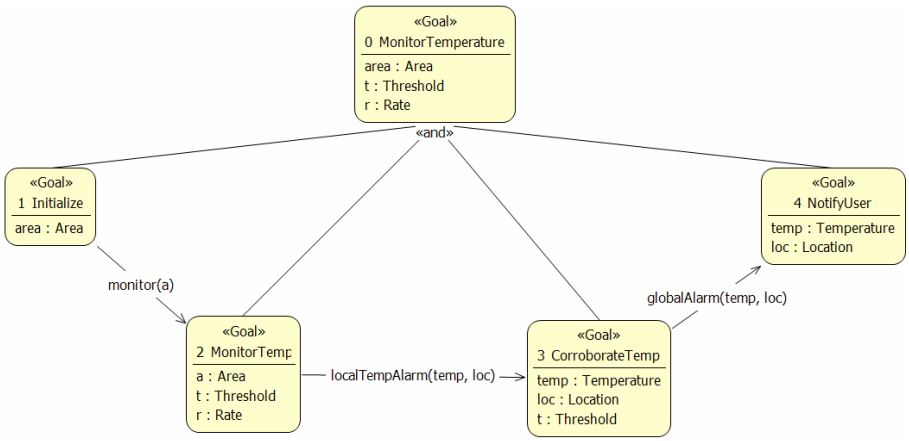
1 capture any sequential constraints among goals

2 determine which goals should be created in response to events that occur at runtime

3 document parameters required to define a unique goal state.

If goal A must be completed before goal B can be pursued, then it is said that goal A *precedes* goal B. As the TMS system operates in parallel, there are no precedence relations in the goal model. New goals are often generated in response to specific events that occur within the environment or system and multiple instances of such goals may be active at any time. In the TMS system, new instances of the CorroborateTemp and NotifyUser goals are created whenever a local alarm or global alarm is raised. When multiple instances of a goal may exist, parameters are used to uniquely define and identify each goal. With a NotifyUser goal, which is created each time a global alarm is raised, a user would need to know the temperature reading as well as the location of the node that raised the alarm.

Figure 7 Refined goal model (see online version for colours)



A GMoDS version of the goal model for the TMS system is shown in Figure 7. Triggers are represented by arrows decorated with an event name and a set of event parameters. When instantiated, the initialise goal is assigned to an agent to determine how many MonitorTemp goals should be created to monitor the entire area. These MonitorTemp goals are assigned to agents who use their sensing capabilities to monitor the temperature. When the sensed temperature exceeds the preset threshold t, the agent raises the localTempAlarm(temp,loc) event that triggers the instantiation of the CorroborateTemp goal. This goal is assigned to an appropriate agent who attempts to corroborate the reading. If it does, the agent raises a globalAlarm(temp,loc) event, which causes the instantiation of a NotifyUser goal. The NotifyUser goal is then assigned to an agent capable of interacting with the user.

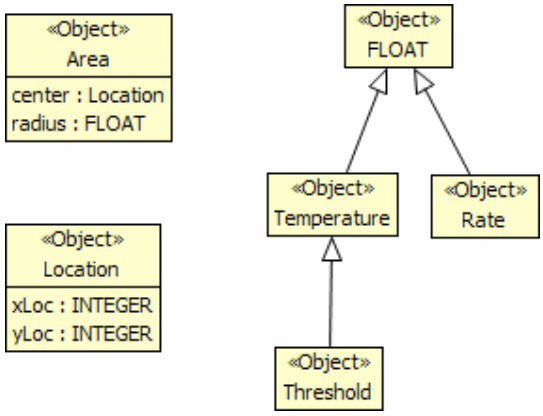
3.4.4 Model domain

The aim of the model domain task is to capture the object types, relationships, and behaviours that define the domain in which agents will sense and act. O-MaSE uses a simple domain model to capture the object types that agents interact and reason about. The domain model captures the environment as a set of object types and agents that are situated in the environment. Object types are defined by a name and a set of attributes. In O-MaSE, domain object types are similar to object classes rather than instances.

The domain model is developed using traditional domain modelling or domain analysis techniques common to many object oriented development methodologies (Prieto-Diaz and Arango, 1991). Object types from the domain model are commonly used to specify goal and event parameters in the goal model, to define message parameters in the protocol model, to specify constraints in the policy model, and to specify the result of agent actions in the action model.

The domain model of the TMS system is shown in Figure 8. As this is a simple system, the model is somewhat small. However, in order to be able to understand the goal model of Figure 7, one must understand the semantics of each attribute and parameter. Thus, the domain model defines the object types temperature, threshold, and rate as base floating point types while area is defined as a circle with a radius. Each location is denoted by an xLoc and yLoc attribute.

Figure 8 Domain model (see online version for colours)



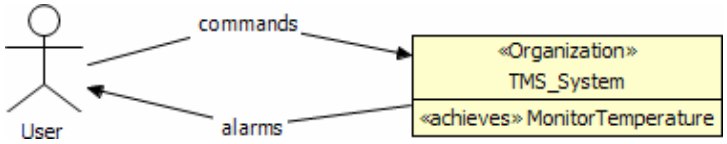
3.4.5 Model organisation interfaces

The objective of the model organisation interfaces task is to identify the organisation's interfaces with external entities, whether they are other agents, organisations, or actors external to the system.

To capture the organisation's interfaces, various classes of external entities are scrutinised to determine if the organisation needs to interact with them. If the organisation is a sub-organisation (an OA) of a higher-level organisation, the interactions between the roles/agents in the higher-level organisation and the OA define the initial set of interactions with this sub-organisation. However, if this is a stand-alone, or top-level organisation, the developer should considers interactions required with users as well as existing systems or databases to find the appropriate interfaces. Once identified, protocols are identified between the organisation and the external entities. There should be a protocol for each type of interaction and thus there can be more than one protocol with a given external entity. The interfaces are defined in an organisation model, which depicts a single organisation interacting with a set of external actors. All external entities are modelled as external actors. The details of the protocols are defined later via the model protocols task.

Figure 9 shows the organisation model for the TMS system. The TMS system is a single-level organisation that interfaces directly with a single user. The user issues controls the system via the commands protocol, while the system provides feedback to the user via the alarms interaction protocol.

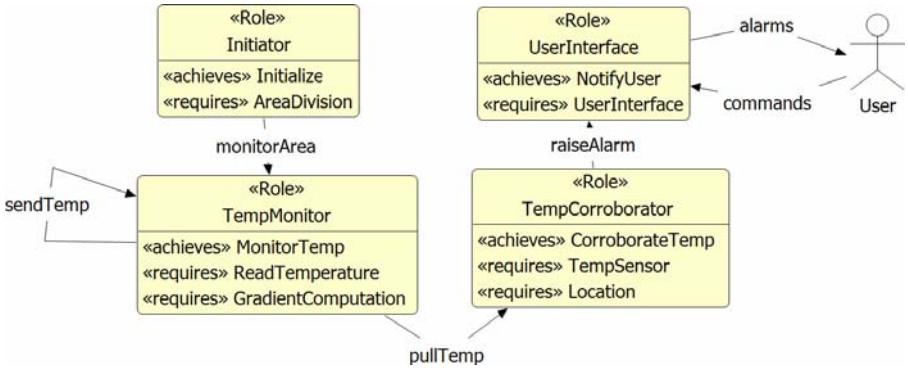
Figure 9 Organisation model (see online version for colours)



3.4.6 Model roles

The model roles task identifies all the roles in the organisation as well as their interactions with each other and with external actors. The result of the model roles task is a role model. The goal of role modelling is to assign each leaf goal from the organisation goal model to a specific role. As a first cut, a single role is often created for each leaf goal. However, it is sometimes beneficial to enable a single role to achieve multiple types of goals. However, it is also true that organisations that are more flexible can be designed by having multiple roles capable of achieving the same type of goal. The designer must also identify interactions between roles as well as with external actors. Interactions with external actors can be derived directly from the organisation model if provided.

Figure 10 Role model (see online version for colours)



The TMS role model in Figure 10 defines four roles, one for each leaf goal in Figure 7: Initiator, TempMonitor, TempCorroborator, and UserInterface. Each role requires various capabilities, which include hardware sensors, such as ReadTemperature, as well as software algorithm such as GradientComputation. Although not stipulated in the model itself, the TempMonitor and TempCorroborator roles are designed to run on remote sensor platforms while the UserInterface and InitiatorRoles can execute on any capable computer. Notice that the user actor defined in the organisation model is also in

the role model. Each role is further defined using either the define roles or the define role goals tasks described next.

3.4.7 Define roles

The purpose of the define roles task is to define the behaviour and capabilities required for an agent to play a role. In addition, constraints may also be specified. In the define roles task, the designer specifies the capabilities required by a role, the goals the role is able to achieve, constraints associated with the role, and the plan(s) that implement the role (If the required capabilities and goals that can be achieved by the roles have already been defined in the role model, these may be omitted). These plans are developed using model plan task as described in Section 3.4.12. The role description document for the TMS system is shown in Table 2. In this case, there is a single plan associated with each role. If a role may be used to achieve multiple goals, then the role may possess multiple plans.

Table 2 Role description document

Role	*Achieves*	*Requires*	*Plan*	*Constraints*
Initiator	Initialise	AreaDivision	DivideArea	None
UserInterface	NotifyUser	UserInterface	ControlSys	None
TempMonitor	MonitorTemp	ReadTemperature GradientComputation	Monitor	None
TempCorroborator	CorroborateTemp	TempSensor Location	Corroborate	None

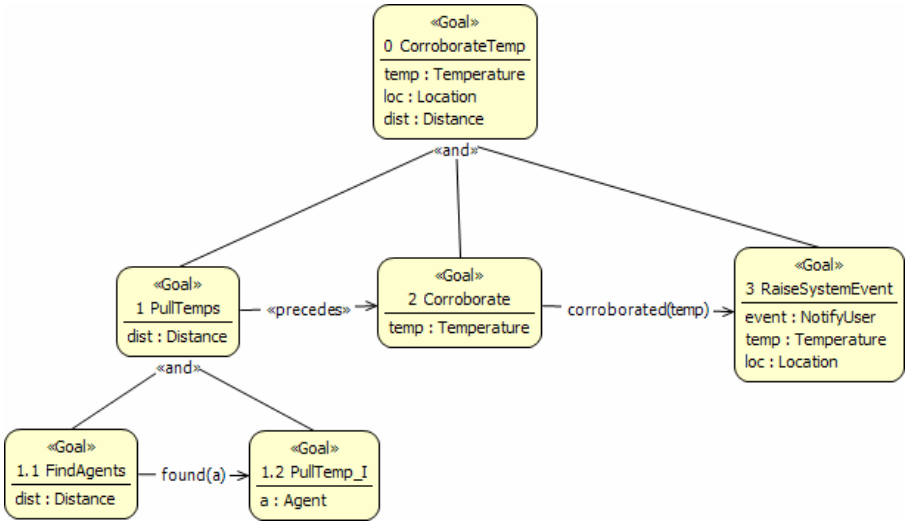
3.4.8 Define role goals

In the define role goals task, role behaviour is defined in terms of a role goal model. The starting point for a role goal model is the leaf goal from the organisation that is to be achieved by the role. Thus, the top goal of a role goal model is a leaf goal from the organisation goal model.

The role goal models have the same semantics as the organisation goal models created with the model goals and refine goals tasks described in Sections 3.4.2 and 3.4.3. In fact, the approach taken to define the goal model is the same as well. The key difference between an organisation goal model and a role goal model is in the level of functionality that can be used to achieve the leaf goals. At the role level, each leaf goal is associated with a capability that can achieve that goal; at the organisation-level, each leaf goal is associated with a role capable of achieving it.

The TempCorroborator role goal model is shown in Figure 11. Precedence is denoted by a ‘precedes’ arrow; in this case, the corroborate goal cannot be pursued until the PullTemps goal has been achieved. When started, the role must pull temperature readings from all agents with a certain distance. Once that is accomplished, it must corroborate its reading against those it has pulled. Finally, if the high temperature reading is corroborated, then the RaiseSystemEvent goal will cause the agent to raise a NotifyUser event at the system level.

Figure 11 TempCorroborator role goal model (see online version for colours)

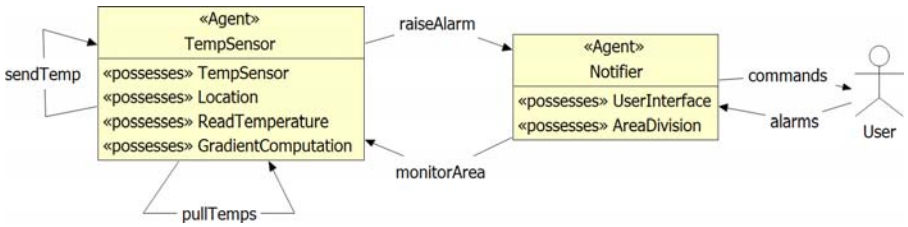


3.4.9 Model agent classes

The model agent classes task identifies the types of agents that may participate in the organisation. Agent classes may be defined to play specific roles, or they may be defined in terms of capabilities, which implicitly define the types of roles that may be played. An agent class is a template for a type of agent in the system. Each agent class identifies the capabilities that it possesses or the roles it can play (or both). In an open system where specific agents are not known *a priori*, an agent class model may not be used as agents register themselves and their capabilities directly with the system; the roles these agents may play is based entirely on the capabilities required for the various roles.

Figure 12 shows an agent class model for the TMS system. As the system consists of homogeneous sensor nodes and a user interface device, there are only two agent types in the system: TempSensor and Notifier. A functioning TempSensor agent is implicitly capable of playing both the TempMonitor and TempCorroborator roles, while a Notifier agent is capable of playing the Initiator and UserInterface roles. Notice that the protocols specified in the role model are inherited by the appropriate agent types in the agent class model and that the user actor is also included.

Figure 12 Agent class model (see online version for colours)

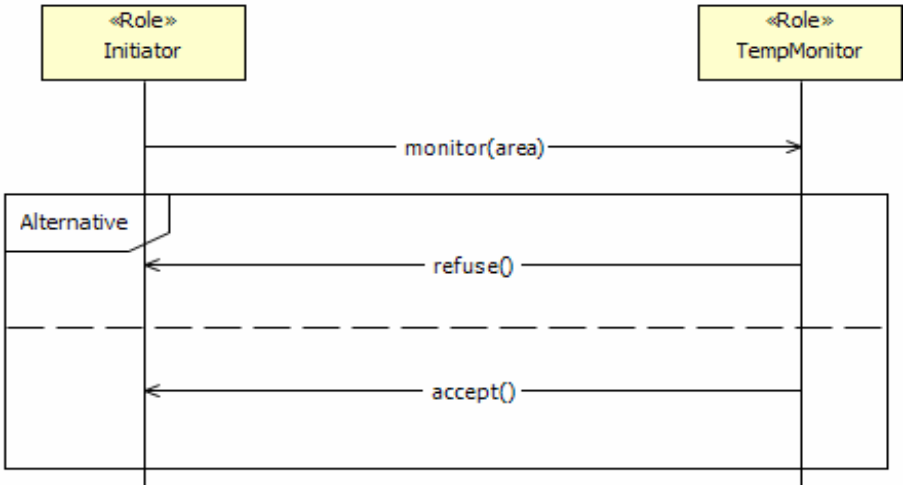


3.4.10 Model protocols

The purpose of the model protocols task is to define the details of the interactions between agents or roles. Since protocols can be specified in organisation models, role models and agent class models, the method engineer may decide which set of protocols to define. If the role model protocols are defined via protocol models, agent classes playing those roles should inherit those protocols. When using aT^3 to design systems, aT^3 provides automated checks to ensure the consistency of these protocols between the various models. The protocol model produced defines the types of messages sent between the two entities and is essentially the same as the AUML (Bauer et al., 2000) and UML (Rumbaugh et al., 2004) interaction models. In each of these models, messages are specified on arrows between lifelines and allowing looping and alternative control flows.

Figure 13 shows the protocol model for the monitorArea protocol in the TMS system. In this case, the initiator sends a monitor(area) request to the TempMonitor. The 'alternative' frame provides an option for the TempMonitor role to return either a refuse() or accept() message.

Figure 13 Protocol model (monitorArea) (see online version for colours)



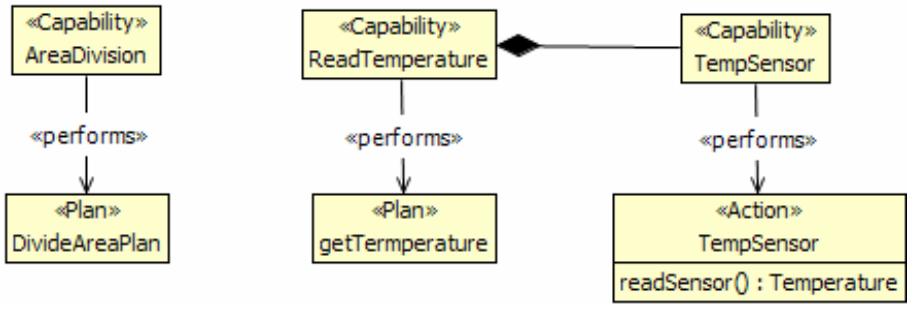
3.4.11 Model capabilities

The model capabilities task is used to define the internal structure of the capabilities possessed by agents in the organisation. The result of the model capabilities task is a capability model. Each capability may be modelled as an action or a plan. An *action* is an atomic functionality possessed by an agent and defined using the model actions task as described in Section 3.4.13. A *plan* is an algorithmic definition (defined via a state machine) of a capability that uses actions and implements protocols. Each plan is defined using the model plans task as presented in Section 3.4.12.

A portion of the capability model for the TMS system is shown in Figure 14. Notice that the AreaDivision capability is represented as a plan, specifically the DivideAreaPlan, while ReadTemperature is represented as a complex capability with a

plan, getTemperature, and a sub-capability, TempSensor. In this case, the getTemperature plan uses the TempSensor by calling its readSensor action.

Figure 14 Capabilities model (see online version for colours)

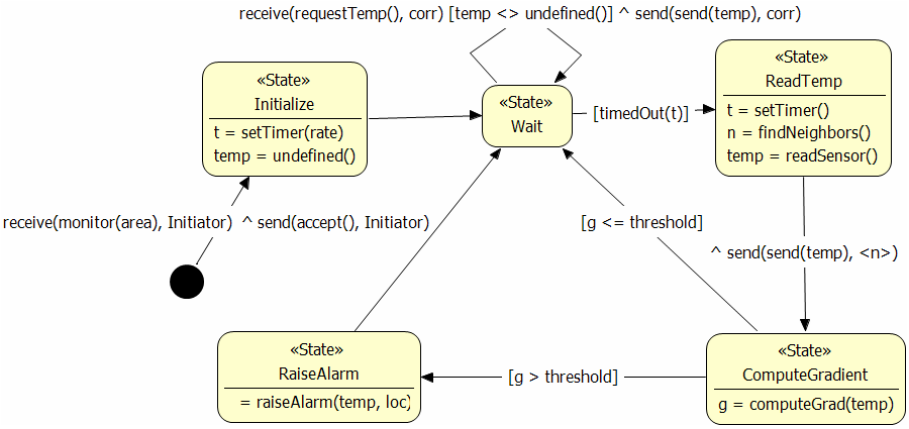


3.4.12 Model plans

The purpose of the model plans task is either to capture how an agent can achieve a specific type of goal using a set of actions (which includes sending and receiving messages) or to define a soft capability. The result of the model plans task is a plan model.

A plan model is specified in terms of a simple finite state machine where states contain action sequences and transitions contain inter-agent communications. Two special actions, send and receive, are used to denote sending and receiving of messages on transitions. User defined actions are carried out sequentially within states. Each action must be defined as part of a capability possessed by the agent performing the plan. Once in a state, the task remains in that state until processing is complete and a transition out of the state is enabled. Variables used in actions and messages are globally visible within the plan.

Figure 15 Plan model (getTemperature) (see online version for colours)



The getTemperature plan (which is part of the ReadTemperature capability) is shown in Figure 15. It is initialised by receiving a monitor message from the initiator. It uses a timer to access its temperature sensor (via the readSensor()action) at the appropriate rate. It then computes the gradient. If the gradient exceeds the threshold it calls raiseAlarm, otherwise, it returns to the wait state. While in the wait state, the plan can respond to requests from corroborator roles to get the current temperature, once it is defined.

3.4.13 Model actions

The model actions task defines the low-level actions used by agents to perform plans and achieve goals. Actions belong to capabilities possessed by agents. Actions are typically defined as a function with a signature and a set of pre and post-conditions. In some cases, actions may be modelled by providing detailed algorithmic information. If using automatic code generation techniques, this information is generally captured as a function or operation in the language being generated. In either case, the action model is usually just a textual document.

readSensor()
Pre: true
Post: readSensor > minTemp $\wedge$ readSensor < maxTemp

In the readSensor example, since the action reads a sensor inputs, there is no precondition and the only guarantee about the output is that it will fall within the advertised sensor range.

3.4.14 Model policies

The model policies task defines a set of formally specified rules that describe how an organisation may or may not behave in particular situations. During the organisation design, the policy modeller captures the desired and/or required properties of the system and writes them in natural language. Once all the policies have been identified, they can be formally specified if needed. For example, the following policy specifies that each TempSensor agent should be assigned to play both the TempMonitor and TempCorroborator roles.

$\forall$a:TempSensor, $\exists$g1:MonitorTemp, g2:CorroborateTemp |
assigned(a, TempMonitor, g1)
$\wedge$ assigned(a, TempCorroborator, g2)

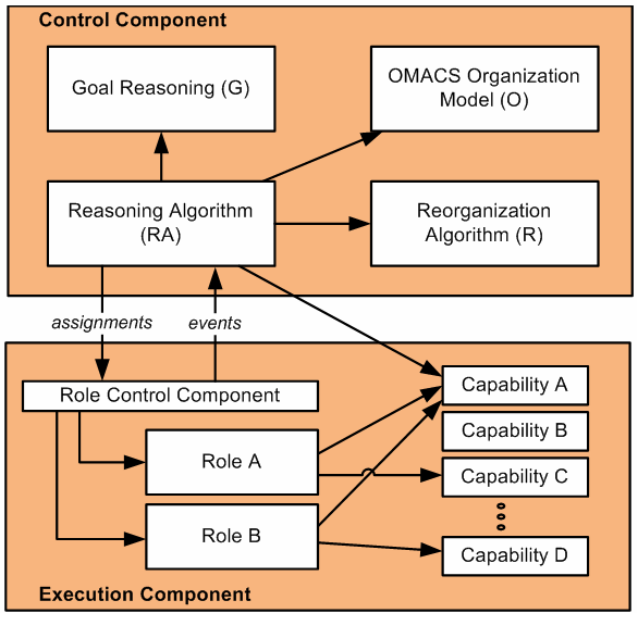
Policies have been used in multi-agent system engineering for some time and several languages, frameworks, enforcement and checking mechanisms have been developed (Bradshaw et. al, 2003; Shoham and Tennenholtz, 1995; Harmon et al., 2007, 2008). In general, policies are used to *restrict* agent behaviour and may be enforced at design time or at runtime. How policies are enforced is a critical decision that affects the way the policy model is used during development. If there is no runtime mechanism designed or provided by the runtime environment, designs and implementations must be evaluated to ensure they conform to the policies.

3.4.15 Generate code

The purpose of the generate code task is to take all the design models created during the development and convert them into code that correctly implements the models. Obviously, there are numerous approaches to code generation based on the runtime platform and implementation language chosen.

The aT^3 toolkit includes an automatic code generation framework. Currently, the only platform supported is JADE (Bellifemine et al., 2007) coupled with our cooperative robotics organisation-based simulator (DeLoach, 2009b). To support OMACS-based systems, the organisation-based agent (OBA) architecture (Figure 16) was created. The *control component* uses XML specifications of the organisation goal, role, and agent models to perform reasoning about goals, the organisation state, and the assignment of agents to roles. The O-MaSE models produced during low level design are used to define the role behaviour in the *execution component*. The OBA architecture supports significant reuse as much of the OMACS reasoning is standard and thus much of the control component code is reusable. A complete description of the architecture can be found in (DeLoach, 2009b).

Figure 16 OBA architecture (see online version for colours)



3.5 Method-roles

Twelve method-roles have been identified as part of the O-MaSE methodology: requirements engineer, goal modeller, domain modeller, organisation modeller, role modeller, agent class modeller, protocol modeller, policy modeller, plan modeller, capabilities modeller, action modeller, and programmer. Each O-MaSE method-role is responsible for carrying out the tasks by applying the appropriate techniques to produce

the work products shown in Table 1. Obviously, this requires the ability to apply the various techniques and to understand the work products that are both inputs to and outputs from those tasks.

3.6 Method construction guidelines

Table 3 shows the method construction guidelines (called process construction guidelines in previously published papers) for the tasks defined in Table 1. These method construction guidelines are defined in terms of a pre-condition and post-condition. The pre-condition specifies the set of work products that must be available prior to the task being undertaken while the post-conditions specify the work products produced by the task. For example, for the model goals task, either a requirements spec must be available or a goal model/GMoDS and a role model must be available. The requirements spec is used when the model goals task is used to model system-level goals while the goal model/GMoDS and role model are used when the task is used to model role-level goals. Disjunctive pre-conditions generally specify alternative ways the Task can be used. However, it does not limit what information can be used in the definition of a model. For instance, the model domain task only requires a requirements spec as input; however, that does not mean that other work products such as goal models cannot be used in the task. This additional information is generally documented in the individual task definitions.

Table 3 Method construction guidelines

Task	*Pre-condition*	*Post-condition*
Requirements specification	True	Requirements spec
Model goals	Requirements spec ∨ ((Goal Model ∨ GMoDS) ∧ Role Model)	Goal model
Refine goals	Goal Model	GMoDS
Model domain	Requirements Spec	Domain model
Model organisation interfaces	Requirements Spec ∧ GMoDS	Organisation model
Model roles	GMoDS ∧ Organisation Model	Role model
Define roles	Role Model	Role description
Model agent classes	GMoDS ∨ Role Model ∨ Organisation Model	Agent class model
Model protocols	Role Model ∨ Agent Class Model	Protocol model
Model policies	GMoDS ∨ Organisation Model ∨ Role Description ∨ Agent Class Model	Policy model
Model plans	(GMoDS ∧ Role Model) ∨ (GMoDS ∧ Agent Class Model)	Plan model
Model capabilities	Role Model ∧ Agent Class Model ∧ Domain Model	Capability model
Model actions	Capability Model ∧ Domain Model	Action model
Code generation	(Plan Model ∨ Protocol Model) ∧ (Capability Model ∨ Action Model)	Source code

Errata shown in red on this page.

aT^3 is based on agentTool 1 and 2, which supported the original MaSE methodology. The original versions of agentTool were written as standalone Java tools that supported graphical model creation, protocol verification, semi-automatic analysis to design transformations, and code generation.

aT^3 is a completely new development and was developed as a set of eclipse plug-ins. Eclipse (http://www.eclipse.org/) is an open-source integrated development environment that supports easy extension through its plug-in-based architecture. Eclipse was chosen as the base for aT^3 due to this extensibility, support for graphic-based editors, and for the ability to create methods, designs, and code within the same environment. In addition, the EPF (http://www.eclipse.org/epf/) provides basic tools that support building custom methods.

In aT^3, there is a separate plug-in for each O-MaSE model and each plug-in accesses a single core plug-in that implements the O-MaSE meta-model. This multi-plug-in architecture supports the goal of allowing O-MaSE to be highly tailorable and extensible. None of the models are required and new models may be incorporated into the tool by adding a new plug-in to create/edit the model and new consistency rules to verify consistency with other models.

The aT^3 development environment actually includes four components that are integrated into a single tool. These components are the graphical editor, the process editor, the verification framework, and a code generation facility. Each component is discussed below.

4 agentTool III

The aT^3 development environment is built on the eclipse platform (Garcia-Ojeda et al., 2009b; DeLoach et al., 2009). The core elements of aT^3 are the model creation tools that support the analysis, design, and implementation of multi-agent systems following the O-MaSE methodology. aT^3 also provides verification and metrics computation components, as well as the ability to compose, verify, and maintain custom O-MaSE complaint methods. The aT^3 project webpage (http://agentTool.cis.ksu.edu/) contains the latest version of aT^3 for download and includes tutorials, documentation, and examples.

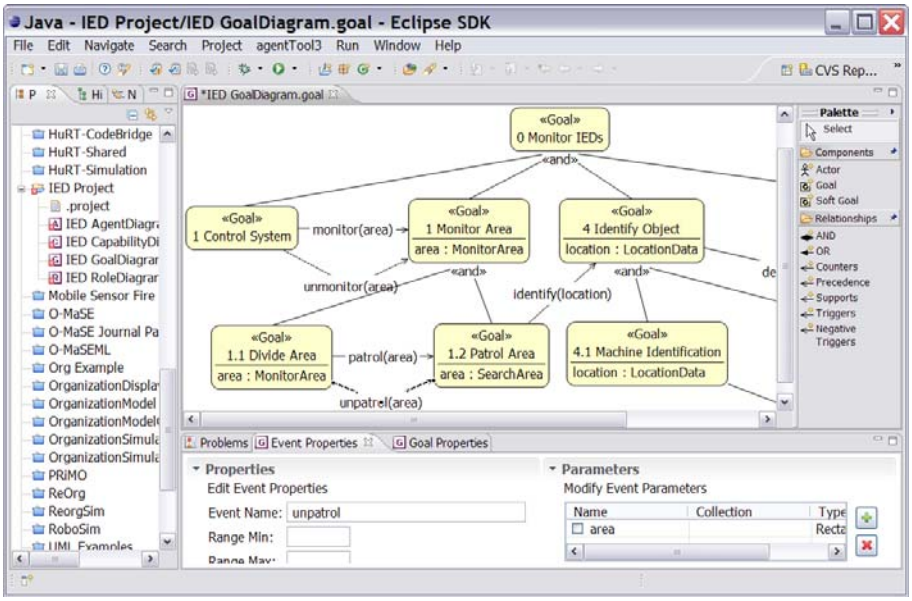
4.1 Graphical editor

The aT^3 graphical editor supports the graphical editing of each of the O-MaSE models described in Section 3.4. These models, when combined, define the design of a MAS using concepts from the O-MaSE meta-model. A designer creates models in aT^3 by dragging model elements from a palette and placing them onto the drawing panel. Built-in validation ensures that only valid connections are made between the appropriate model elements. To edit the internal details of model elements, aT^3 also provides pop-up panels for items such as agent attributes and event parameters.

A screenshot of aT^3 is shown in Figure 17. On the left side of the screen, the eclipse package explorer allows the user to organise and store O-MaSE models in projects. Generally, subdirectories within projects refer to sub-organisations in the system design, thus the package explorer file structure mimics the hierarchical structure of the system. The model shown is an agent class diagram. The icons shown in the palette on the right side of the screen show the valid model elements and relations that may be added to the

model. To add a model element to the model, users simply click on the model element in the palette and then click where they want to place the model element in the model. Once the model element has been placed in the model, it may be edited or moved to another location. The protocol model elements are slightly different in that they are added between two actors or agents. To add a protocol, the user first clicks on the protocol icon in the palette and then on the two actors/agents that participate in the protocol. After placing the protocol, the name may be edited. To add relationships between model elements, the user also clicks on the desired relationship in the palette and then click on two model elements already in the model. Relationships have fixed names that may not be edited.

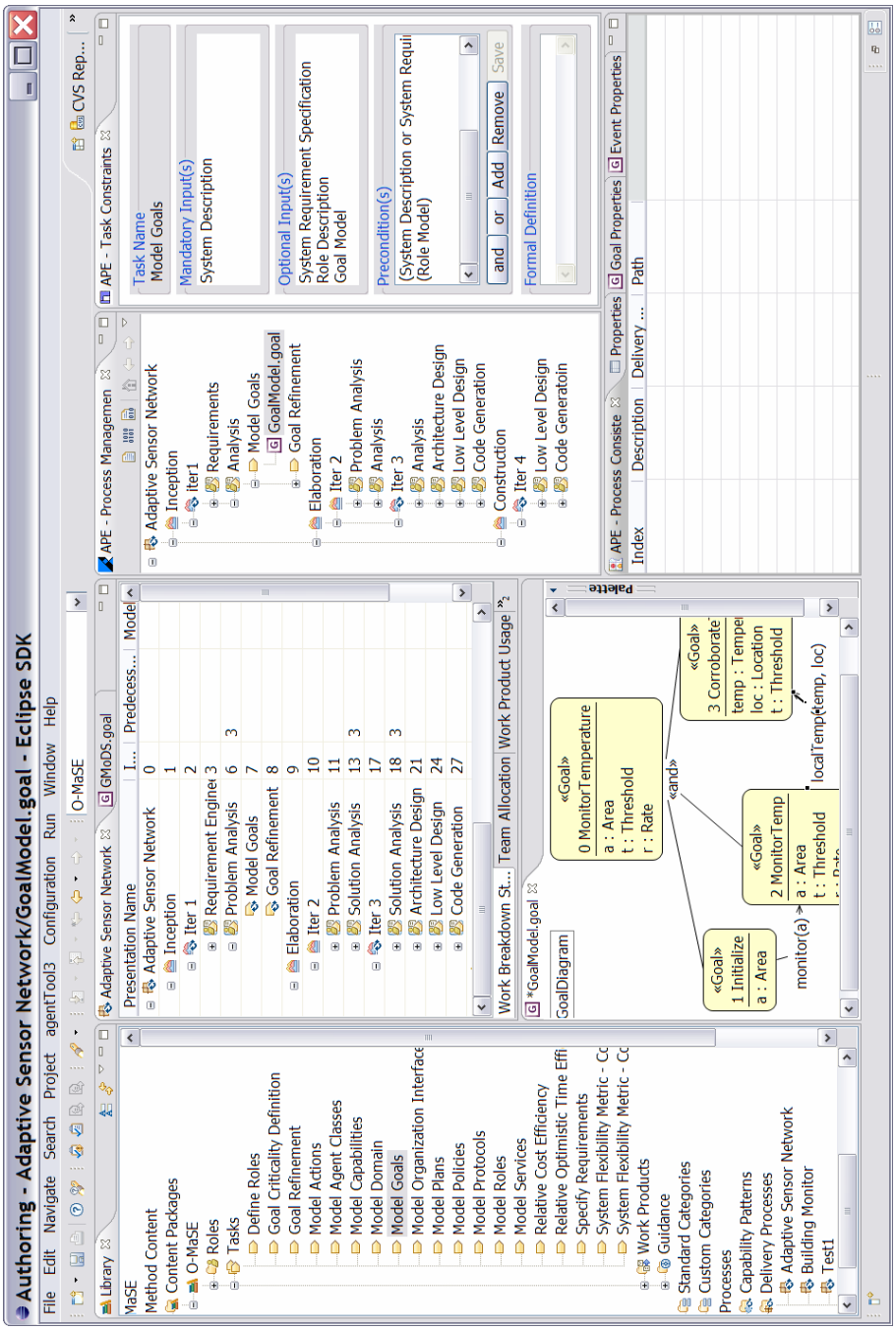
Figure 17 aT^3 graphical editor (see online version for colours)



4.2 Process editor

The APE is based on the EPF and allows method engineers to compose O-MaSE compliant methods (Garcia-Ojeda et al., 2009a). APE provides five basic structures: a method fragment library, the process editor, a set of task constraints, a process consistency checker, and a process management tool as shown in Figure 18. The *library* is a repository of O-MaSE compliant method fragments, which can be extended by APE users. The *process editor* allows users to create and maintain O-MaSE compliant methods. The *task constraints* view helps method engineers specify method construction guidelines to constrain how tasks can be assembled, while the *process consistency* mechanism verifies the consistency of custom methods against those constraints. Finally, the *process management tool* provides a way to measure project progress using earned value analysis. For more details, see Garcia-Ojeda et al. (2009a).

Figure 18 agentTool process editor (see online version for colours)



4.3 Verification framework

The aT^3 verification framework gives designers a way to maintain consistency between their O-MaSE models using a predefined set of rules. Since methods are customised, this rule set can also be customised by turning on and off certain rules. Each time a model is saved, the verification framework checks that document against all related documents in the current project using the currently enabled rules. Verification problems are shown to the user through the eclipse problems panel similar to compiler errors and warnings as shown in Figure 17.

4.4 Code generation facility

Automatic code generation is also available in aT^3. Currently, the only platform targeted has been JADE (Bellifemine et al., 2007). However, a framework has been created consisting of the organisation, operation, social, and environment levels. At the organisation level, agents and roles are chosen for achieving specific goals. At the operation level, agents achieve goals by performing actions based on their available capabilities. At the social level, agent's interactions are captured via messaging, while at the environment level, the knowledge of object types and relationships are generated. Due to the detail of the O-MaSE models, the aT^3 JADE generator is capable of generating 100% of the code necessary to create functional JADE systems. The generated code relies on pre-written Java code for each action specified in the action model.

5 Examples

In order to demonstrate our approach to assembling customised methods using O-MaSE, two examples deriving custom O-MaSE methods are presented. Readers can find applications of O-MaSE in other fields such as information systems (DeLoach et al., 2008), robotics (Garcia-Ojeda et al., 2008), and cooperative software agents (Garcia-Ojeda et al., 2009a). The first example is an adaptive sensor network (ASN) that, while highly adaptive, is computationally expensive. The second example is a much more straightforward sensor-based building monitoring system (BMS) whose operation relies on relatively simple sensors with little computational overhead.

5.1 Adaptive sensor networks

The first example is the development of an ASN system. The ASN is designed to be able to detect and track vehicles moving over a large area. Multiple sensor types will be deployed including motion detectors, magnetometers, and heat detectors. In addition, special radiation sensors will be deployed to determine if any vehicles are radioactive. Sensors will be deployed in overlapping patterns based on the probability of vehicles actually being in that area. To maximise battery life, sensors will be turned off as much as possible and only awakened when needed. Generally, a few motion detector sensors will be on to detect possible vehicles. When a vehicle is detected, additional sensors will be turned on to verify its location and track the vehicle as it moves.

Therefore, an ASN system must be able respond to specific events that occur in the environment as well as be able to reason about individual sensor capabilities and

re-organise to achieve the desired system functionality. This highly adaptive behaviour is exactly what OMACS systems are designed to achieve and thus the system must be designed to include all OMACS required entities as shown in Figure 3.

Figure 19 ASN method (see online version for colours)

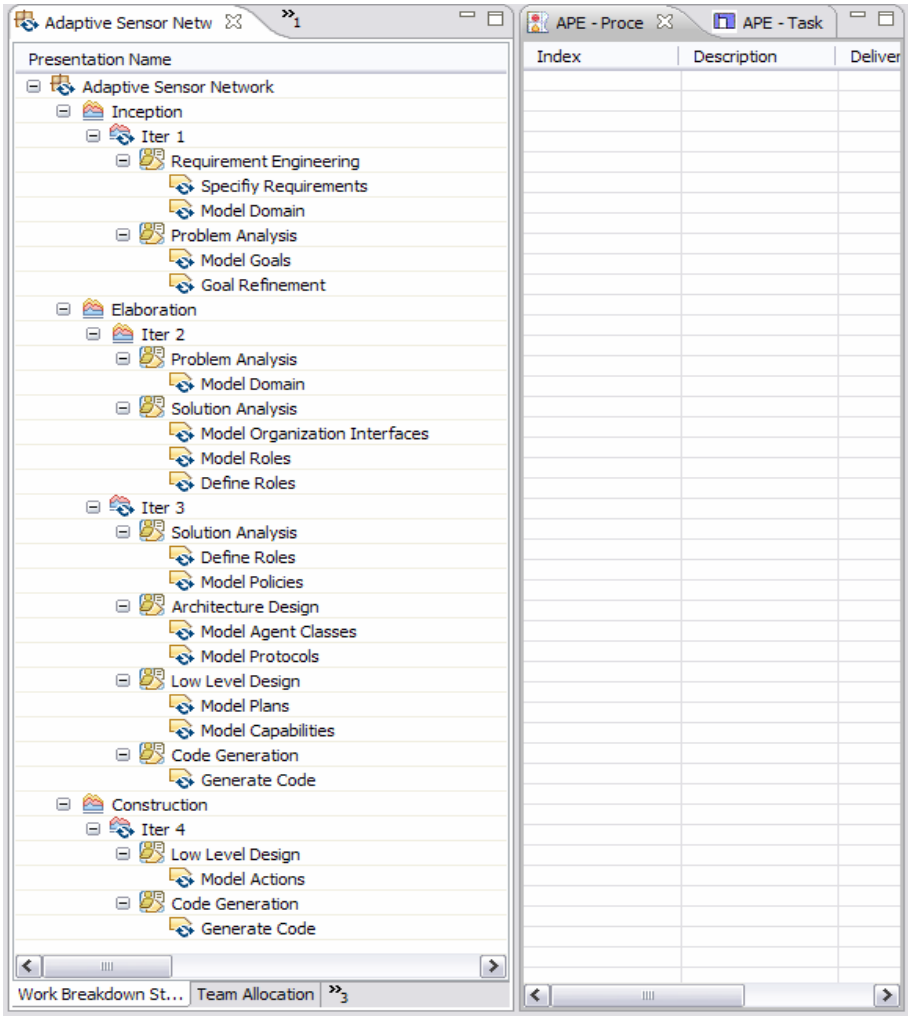


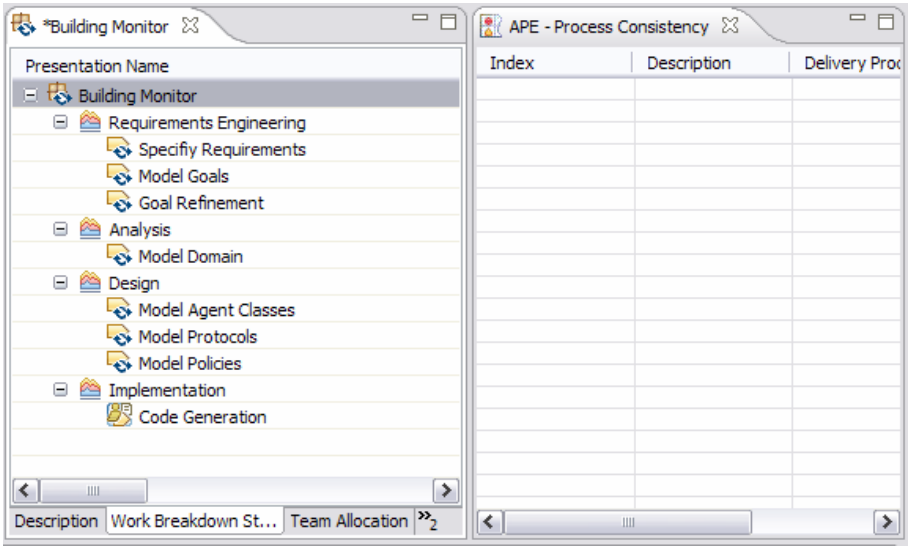
Figure 19 shows an overview of the method for developing an ASN. As shown, the method uses an iterative approach with three phases (inception, elaboration, and construction) and four iterations. Since the ASN systems requires an OMACS-based approach, it is necessary to ensure that all OMACS entities are produced as work products. In general, an OMACS-based method should produce a goal model, role model, agent class model, and policy model. The additional domain, organisation, protocol, capability, plan, and action models were added to support development of the base models or to support code generation. The right side of Figure 19 shows the process

consistency checker for the APE. As shown, there are no inconsistencies and thus the method is O-MaSE compliant. As this method is designed to produce a highly adaptive, OMACS-based system, 14 out of the 15 O-MaSE tasks, or 93%, are used to define the method. As demonstrated here, methods used to create systems that are more complex tend to be larger in terms of number of O-MaSE tasks required.

5.2 Building monitoring system

The second example is also taken from sensor network domain to illustrate the flexibility of O-MaSE within a single domain. In this case, a BMS will be developed. In this case, the BMS will have a predefined set of sensor types and each sensor will be deployed to a fixed location. Each sensor will be hardwired to the building's electrical supply, so power consumption is not an issue. Each sensor will sense at regular intervals and send its data to its neighbours for verification. All verified data will be sent via a predefined path to a central computer. Each sensor will be modelled as an agent that achieves a specific set of goals; the system will not need to reason about its capabilities or reorganise. While another contractor will design the internal operation of the individual agents, each agent will have to conform to system specific policies to ensure compatibility between agents.

Figure 20 Building monitoring system (see online version for colours)



Given the well-defined nature of the system, a straightforward waterfall development approach is chosen. Appropriate models are selected based on the implementation needs. Because there is no need for adaptivity in terms of reassigning agent responsibility, an organisational approach is not required. Therefore, a straightforward agent-centred approach is taken where agents are designed to achieve specific goals. Figure 20 shows the method developed for the project. The method consists of three different phases: requirements engineering, analysis, design, and implementation. In this case, only eight out of 15, or 53% of tasks defined by O-MaSE are used in this method. When compared

to the previous example, this method is much simpler and thus more appropriate to this specific system development. Again, the right side of Figure 20 shows that the method is O-MaSE compliant.

6 Related work

This section provides a comparison of O-MaSE against several other well-known agent-oriented software engineering methodologies in three different categories: process features, model features, and supportive features. These categories are taken from the evaluation of the APSECS methodology when compared to PASSI, INGENIAS, ANEMONA, Gaia, ROADMAP, Tropos, Prometheus, and ADELFE (Cossentino et al., 2009). ASPECS is a modern agent-oriented methodology focused on complex, organisation-based system following a holonomic perspective; it is similar in many aspects to O-MaSE as is discussed below. Instead of reproducing a complete evaluation here (most of which would essentially duplicate the ASPECS evaluation), a discussion of the unique aspects of O-MaSE is provided as related to these categories and the other methodologies. For the complete evaluation of the other methodologies, the reader is referred to Section 6 of Cossentino et al. (2009), which includes evaluations of the aforementioned methodologies with respect to the same three categories. The three basic categories were derived from the four categories used by Tran and Low (2005). Each category is defined below along with a discussion of the unique aspects of O-MaSE as compared to the other methodologies.

6.1 Process features

The process features category attempts to judge generality and completeness of a methodology. Questions used to evaluate methodologies in this category include

1 standard lifecycle(s) supported

2 standard development activities included

3 whether or not the methodology is domain dependent or independent.

Each of the methodologies studied claim to allow iterative and incremental lifecycles based on modern approaches. However, the methodologies generally fail to specify the exact relationships between the various activities that would allow activities to be placed appropriately in varying iterations. Support is available for methodologies whose processes have been formally defined such as INGENIAS, PASSI, and O-MaSE.

Most of the modern methodologies considered cover the entire development lifecycle from requirements through design and implementation and/or deployment. Only the early methodologies such as Gaia and Tropos cover only analysis and design activities. A unique aspect of O-MaSE is the ease with which it can be extended. Due to its inherent design as a set of fragments, new tasks and models may be easily added without causing changes to existing methods. For example, related work on design metrics for OMACS-based systems resulted in the addition of several additional tasks and work products to current version of O-MaSE as released in aT^3 such as system flexibility (Robby et al., 2006).

6.2 Model features

The model features category attempts to judge the focus of the process models and their completeness in terms of handling non-agent concepts. Criteria used to evaluate methodologies in this category include

1 agent focused versus organisation focused

2 support for levels of system decomposition

3 support for modelling interactions with the environment

4 support for modelling of domain knowledge

5 formal foundation and semantics.

In terms of focus, several of the earlier methodologies such as PASSI, Prometheus, and ADELFE are agent focused, while the newer methodologies tend to be organisation focused. O-MaSE appears to be unique in its support for both points of view as demonstrated in the example methods of Sections 5.1 and 5.2. As Cossentino et al. (2009) point out, within the organisation-based approaches, some focus on concepts of agents, roles, and groups while other highlight norms, which moves toward the concepts of electronic institutions (Noriega and Sierra, 2002). Again, O-MaSE shows its flexibility by supporting either approach.

In terms of modelling complex systems via levels of system decomposition, only the holonomic methodologies, ANEMONA and ASPECS provide such support. Here, O-MaSE provides a hierarchical decomposition approach using agent organisations (AOs), which are closely related to holarchies.

O-MaSE also provides support for modelling of the environment and interactions with the environment. One of the main purposes of the O-MaSE domain model is for capturing object types and their relationships in the environment while actions allow developers to specify the effects of agent operations on those environment objects. Because protocols may be specified in terms of actions (as well as messages), complex interactions with the environment may also be modelled.

While O-MaSE does not use formal notations except in the case of policies, the formalisation of its meta-model does allow the use of formal model checking techniques to provide *predictive metrics*. For instance, Robby et al. (2006) describe a system *flexibility* metric that measures how many different ways an OMACS-based system can achieve its overall goal. Automated techniques for computing such metrics have been incorporated into aT^3.

6.3 Supportive features

Finally, the supportive features category looks at the methodology's support for standards, tools, and complex system concepts. Criteria used to evaluate methodologies in this category include

1 tool and library support

2 support for open versus closed agent systems

3 support for dynamic, self-organising, and reconfiguring systems.

A distinguishing aspect of O-MaSE is its tool support. As discussed above, aT^3 provides and integrated environment (with eclipse) that supports method engineering, model development and verification, design level predictive metrics, and automatic and manual code generation. Because it is based on the eclipse plug-in approach, aT^3 is extremely extensible; new method fragments, new models, and new code generation, deployment, and testing tools can be added by adding new plug-ins. The INGENIAS IDK also supports similar aspects although the process editor is not fully integrated with IDK.

Cossentino et al. (2009) point out that APSECS is "the only process that supports both open and dynamic systems and merges an agent-oriented approach with a knowledge-engineering approach". Clearly, O-MaSE also supports these areas. While ASPECS focuses more heavily on the use of domain knowledge by specific references to its ontology, O-MaSE, being based on OMACS, has a more precisely defined mechanism to support dynamic, reconfigurable open systems. In addition, the second main use of the O-MaSE domain model is to capture ontologies in support of open systems.

7 Conclusions and future work

The O-MaSE methodology framework integrates a suite of technologies aimed at removing impediments to the industrial acceptance of agent technology. O-MaSE provides a customisable agent-oriented methodology based on consistent, well-defined concepts supported by plug-ins to an industrial strength development environment. The O-MaSE methodology framework allows developers to create custom agent-oriented methods using a set of well-defined method fragments that support a variety of system types and complexities. This is achieved in O-MaSE via the O-MaSE meta-model, a set of method fragments, and a set of method construction guidelines. Each aspect of the O-MaSE methodology framework is supported by the aT^3 integrated development environment, which supports method creation and maintenance, model creation and verification, and code generation and maintenance.

The main advantages of this approach are

1 O-MaSE supports agent-centred, organisation centred, closed or open agent systems, based on the method fragments used in an appropriate custom method

2 each O-MaSE method fragment is defined over a common meta-model that also directly supports complex adaptive systems based on the OMACS organisation model and its associated architectures and algorithms

3 the O-MaSE method construction guidelines define how method fragments may be combined in to assemble O-MaSE compliant methods

4 O-MaSE is fully support by aT^3, which supports the creation and implementation of O-MaSE compliant method as well as supporting the creation and verification of systems using those methods.

Because O-MaSE and aT^3 provide a comprehensive environment for developing multi-agent and organisation-based systems, it also provides an excellent platform for additional research and development. We plan to continue investigating formal compositional approaches for building complex, adaptive systems using semi-automatic

design-time metrics as well as automatic runtime composition using O-MaSE models (Oyenan et al., 2009). This work will be integrated by extending O-MaSE with new method fragments as well as adding new functionality in aT^3.

A very important area that should be investigated further is the integration of O-MaSE concepts with other MAS meta-models, which have been the subject of much research (Azaiez et al., 2006; Bernon et al., 2005; Beydoun et al., 2009). When development of O-MaSE started in 2005 (DeLoach, 2006), there were no meta-models that captured the key elements required to support OMACS-based systems, namely a direct relation between agents, roles, goals, and capabilities. Thus, the O-MaSE meta-model was developed in parallel with recently published meta-models. However, new work in synthesising common MAS meta-models, specifically the FAML meta-model (Beydoun et al., 2009), provides great promise of producing a general meta-model capable of supporting standardisation of concepts across the agent community. While FAML does not currently support all the required concepts and relations to support an OMACS-based systems (there is currently no notion of capabilities at design or runtime that are possessed by agents and required to play specific roles), the extensibility of FAML has been shown. Future work on O-MaSE should include a detailed study of extending more general meta-model such as FAML to replace the O-MaSE meta-model. Such use of common meta-models would significantly enhance the overall goal of many MAS researchers of combining reusable method fragments from multiple MAS methodologies.

Another area of future work would be to recast O-MaSE in terms of ISO/IEC 24744, the SEMDM. Using SEMDM as the basis of O-MaSE would allow a more precise description of the relationship between the modelling of methodology elements and their instances during methodology enactment. It would also allow a more precise description of the relationships between the process and products currently captured in the O-MaSE method construction guidelines via the SEMDM action element. Finally, SEMDM would allow for a more precise definition of how the O-MaSE (or extended FAML) meta-model and modelling notations relate to specific work products.

We are also interested in approaches for dealing with human agents as part of the system. We are currently studying how humans and agents can exist together within multi-agent teams. We are looking at extending the OMACS model and thus, by extension the O-MaSE (or an extended version of FAML) meta-model. Clearly, such an extension should ensure backward compatibility and will likely require the integration of new method fragments into O-MaSE.

We are also investigating how to model and reason about agent interactions at runtime. Again, this will likely require introducing new concepts into the O-MaSE (or FAML) meta-model, such as first-class interactions and interaction goals, as well as providing new models for capturing such information.

Acknowledgements

This work was supported by grants from the US National Science Foundation (0347545) and the US Air Force Office of Scientific Research (FA9550-06-1-0058 and FA9550-09-1-0108).

References

Azaiez, S., Huget, M.P. and Oquendo, F. (2006) 'An approach for multi-agent metamodelling', *Multiagent and Grid Systems*, Vol. 2, No. 4, pp.435–454.

Bakshi, A., Prasanna, V.K., Reich, J. and Larner, D. (2005) 'The abstract task graph: a methodology for architecture-independent programming of networked sensor systems', *Workshop on End-to-End, Sense-and-Respond Systems, Applications, and Services (EESR 05)*, 5 June 2005.

Bauer, B., Müller, J. and Odell, J. (2000) 'Agent UML: a formalism for specifying multiagent software systems', in Ciancarini, P. and Wooldridge, M. (Eds.): *Agent-Oriented Software Engineering: Proceedings of the First International Workshop (AOSE-2000)*, Springer, Berlin.

Bellifemine, F.L., Caire, G. and Greenwood, D. (2007) *Developing Multi-agent Systems with JADE*, Wiley & Sons, England.

Bernon, C., Cossentini, M. and Pavon, J. (2005) 'Agent-oriented software engineering', *The Knowledge Engineering Review*, Vol. 20, No. 2, pp.99–116.

Beydoun, G., Gonzalez-Perez, C., Low, G. and Henderson-Sellers, B. (2005) 'Synthesis of a generic MAS meta-model', *SIGSOFT Softw. Eng. Notes*, Vol. 30, No. 4, pp.1–5.

Beydoun, G., Low, G., Henderson-Sellers, B., Mouratidis, H., Gomez-Sanz, J.J., Pavon, J. and Gonzalez-Perez, C. (2009) 'FAML: a generic meta-model for MAS development', *IEEE Trans. Softw. Eng.*, Vol. 35, No. 6, pp.841–863.

Blair, G., Bencomo, N. and France, R.B. (2009) 'Models@ run.time', *Computer*, Vol. 42, No. 10, pp.22–27.

Bradshaw, J., Uszok, A., Jeffers, R., Suri, N., Hayes, P., Burstein, M., Acquisti, A., Benyo, B., Breedy, M., Carvalho, M., Diller, D., Johnson, M., Kulkarni, S., Lott, J., Sierhuis, M. and Hoof, R.V. (2003) 'Representation and reasoning for DAML-based policy and domain services in KAoS and Nomads', *AAMAS '03: Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, ACM Press, New York, NY, USA, pp.835–842.

Brinkkemper, S. (1996) 'Method engineering: engineering of information systems development methods and tools', *Information and Software Technology*, Vol. 38, No. 4, pp.275–280.

Calisti, M. and Rimassa, G. (2009) 'Opportunities to support the widespread adoption of software agent technologies', *International Journal of Agent-Oriented Software Engineering*, Vol. 3, No. 4, pp.411–415.

Castro, J., Kolp, M. and Mylopoulos, J. (2002) 'Towards requirements-driven information systems engineering: the Tropos project', *Information Systems*, Vol. 27, No. 6, pp.365–389.

Coelho, R., Kulesza, U., von Staa, A. and Lucena, C. (2006) 'Unit testing in multi-agent systems using mock agents and aspects', *Proceedings of the 2006 international Workshop on Software Engineering for Large-Scale Multi-Agent Systems*, Shanghai, China, 22–23 May 2006, ACM, New York.

Cossentino, M., Gaglio, S., Garro, A. and Seidita, V. (2007) 'Method fragments for agent design methodologies: from standardisation to research', *International Journal of Agent-Oriented Software Engineering*, Vol. 1, No. 1, pp.91–121.

Cossentino, M., Gaud, N., Hilaire, V., Galland, S. and Koukam, A. (2009) 'ASPECS: an agent-oriented software process for engineering complex systems', *Journal of Autonomous Agents and Multiagent Systems Systems*, Vol. 20, No. 2, pp.260–304.

DeLoach, S.A. (2006) 'Multiagent systems engineering of organization-based multiagent systems', in Garcia, A., Choren, R., Lucena, L., Giorgini, P., Holvoet, T. and Romanovsky, A. (Eds.): *Software Engineering for Multi-Agent Systems IV*, LNCS Vol. 3914, pp.109–125, Springer, Berlin.

DeLoach, S.A. (2009a) 'Moving multiagent systems from research to practice', *International Journal of Agent-Oriented Software Engineering*, Vol. 3, No. 4, pp.378–382.

DeLoach, S.A. (2009b) 'OMACS: a framework for adaptive, complex systems', in Dignum, V. (Ed.): *Multi-Agent Systems: Semantics and Dynamics of Organizational Models*, pp.76–104, IGI, Hershey, PA.

DeLoach, S.A. and Miller, M. (2010) 'A goal model for adaptive complex systems', *International Journal of Computational Intelligence: Theory and Practice*, Vol. 5, No. 2, (in press).

DeLoach, S.A. and Valenzuela, J.L. (2007) 'An agent-environment interaction model', in Padgham, L. and Zambonelli, F. (Eds.): *AOSE VII/AOSE 2006*, LNCS Vol. 4405, Springer, Heidelberg.

DeLoach, S.A. and Wood, M. (2001) 'Developing multiagent systems with agentTool', in Castelfranchi, C. and Lesperance, Y. (Eds.): *Intelligent Agents VII: Agent Theories Architectures and Languages, 7th International Workshop (ATAL 2000, Boston, MA, USA, 7–9 July 2000)*, LNCS Vol. 1986, Springer, Berlin.

DeLoach, S.A., Oyenan, W. and Matson, E.T. (2008) 'A capabilities based model for artificial organizations', *Journal of Autonomous Agents and Multiagent Systems*, Vol. 16, No. 1, pp.13–56.

DeLoach, S.A., Padgham, L., Perini, A., Susi, A. and Thangarajah, J. (2009) 'Using three AOSE toolkits to develop a sample design', *International Journal of Agent Oriented Software Engineering*, Vol. 3, No. 4, pp.416–476.

DeLoach, S.A., Wood, M.F. and Sparkman, C.A. (2001) 'Multiagent systems engineering', *The International Journal of Software Engineering and Knowledge Engineering*, Vol. 11, No. 3, pp.231–258.

Ferber, J. and Gutknecht, O. (1998) 'A meta-model for the analysis and design of organizations in multi-agent systems', *Proceedings of the 3rd international Conference on Multi Agent Systems*, IEEE Computer Society, Washington, DC.

Ferber, J., Gutknecht, O. and Michel, F. (2003) 'From agents to organizations: an organizational view of multi-agent systems', in Giorgini, P., Müller, J.P. and Odell, J. (Eds.): *Agent-Oriented Software Engineering IV*, LNCS Vol. 2935, pp.214–230, Springer, Berlin.

Firesmith, D. and Henderson-Sellers, B. (2002) *The OPEN Process Framework: An Introduction*, Addison-Wesley, Harlow, UK.

Fuentes-Fernández, R., Gómez-Sanz, J.J. and Pavón, J. (2009) 'Requirements elicitation and analysis of multiagent systems using activity theory', *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans*, Vol. 39, No. 2, pp.282–298.

Garcia-Ojeda, J.C., DeLoach, S.A. and Robby (2009a) 'agentTool III: from process definition to code generation', in Decker, K., Sichman, J.S., Sierra, C. and Castelfranchi, C. (Eds.): *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, 10–15 May 2009, Budapest, Hungary, pp.1393–1394.

Garcia-Ojeda, J.C., DeLoach, S.A. and Robby (2009b) 'agentTool process editor: supporting the design of tailored agent-based processes', *Proceedings of the 24th Annual ACM Symposium on Applied Computing*, 8–12 March 2009, Honolulu, Hawaii, USA.

Garcia-Ojeda, J.C., DeLoach, S.A., Robby, Oyenan, W.H. and Valenzuela, J. (2008) 'O-MaSE: a customizable approach to developing multiagent development processes', in Luck, M. (Ed.): *Agent-Oriented Software Engineering VIII: The 8th Intl. Workshop on Agent Oriented Software Engineering*, LNCS Vol. 4951, pp.1–15, Springer, Berlin.

Georgeff, M. (2009) 'The gap between software engineering and multi-agent systems: bridging the divide', *International Journal of Agent-Oriented Software Engineering*, Vol. 3, No. 4, pp.391–396.

Giorgini, P., Mylopoulos, J. and Sebastiani, R. (2005) 'Goal-oriented requirements analysis and reasoning in the Tropos methodology', *Engineering Applications of Artificial Intelligence*, Vol. 18, No. 2, pp.159–171.

Gonzalez-Perez, C. and Henderson-Sellers, B. (2006) 'A powertype-based metamodelling framework', *Software and Systems Modeling*, Vol. 5, No. 1, pp.72–90.

Harmon, S.J., DeLoach, S.A. and Robby (2007) 'Trace-based specification of law and guidance policies for multiagent systems', *Engineering Societies in the Agents World VIII*, LNCS Vol. 4995, pp.333–349, Springer, Berlin.

Harmon, S.J., DeLoach, S.A., Robby and Caragea, D. (2008) 'Leveraging organizational guidance policies with learning to self-tune multiagent systems', *The Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, 20–24 October 2008, Venice, Italy.

Henderson-Sellers, B. (2005) 'Creating a comprehensive agent-oriented methdology: using method engineering and the OPEN metamodel', in Henderson-Sellers, B. and Giorgini, P. (Eds.): *Agent-Oriented Methodologies*, pp.368–397, Idea Group, Hershey, PA.

Holland, O. and Melhuish, C. (1999) 'Stigmergy, self-organization, and sorting in collective robotics', *Artificial Life*, Vol. 5, No. 2, pp.173–202.

Horling, B. and Lesser, V. (2004) 'A survey of multi-agent organizational paradigms', *Knowl. Eng. Rev.*, Vol. 19, No. 4, pp.281–316.

ISO/IEC (2007) *ISO/IEC 24744, Software Engineering – Metamodel for Development Methodologies*, International Organization for Standardization/International Electrotechnical Commission.

Jennings, N.R., Sycara, K. and Wooldridge, M. (1998) 'A roadmap of agent research and development', *Journal of Autonomous Agents and Multi-Agent Systems*, Vol. 1, No. 1, pp.7–38.

Kroll, P. and Kruchten, P. (2003) *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP*, Addison-Wesley, Reading, MA.

Kruchten, P. (2000) *The Rational Unified Process, An Introduction*, 2nd ed., Addison-Wesley, Reading, MA.

Lam, D.N. and Barber, K.S. (2005) 'Debugging agent behavior in an implemented agent system' in Bordini, R.H., Dastani, M.M., Dix, J. and El Fallah Seghrouchni, A. (Eds.): *PROMAS 2004*, LNCS 3346, pp.104–125, Springer, Heidelberg.

Low, G., Beydoun, G., Henderson-Sellers, B. and Gonzalez-Perez, C. (2009) 'Towards method engineering for multi-agent systems: a validation of a generic MAS meta-model', in Ghose, A., Governatori, G. and Sadananda, R. (Eds.): *Agent Computing and Multi-Agent Systems: 10th Pacific Rim international Conference on Multi-Agent Systems, PRIMA 2007, 21–23 November 2007, Bangkok, Thailand*, LNAI Vol. 5044, Springer, Berlin.

Luck, M., McBurney, P., Shehory, O. and Willmott, S. (2005) *Agent Technology: Computing as Interaction (A Roadmap for Agent Based Computing)*, Southampton, UK, AgentLink.

Molesini, A., Denti, E., Nardini, E. and Omicini, A. (2009) 'Situated process engineering for integrating processes from methodologies to infrastructures', *Proceedings of the 2009 ACM Symposium on Applied Computing (Honolulu, Hawaii)*, ACM, New York, pp.699–706.

Nguyen, D.C., Perini, A. and Tonella, P. (2008) 'A goal-oriented software testing methodology', *8th International Workshop on Agent-Oriented Software Engineering, AOSE 2007*, LNCS Vol. 4951, Springer, Berlin.

Noriega, P. and Sierra, C. (2002) 'Electronic institutions: future trends and challenges', in Klusch, M. Ossowski, S. and Shehory, O. (Eds.): *Cooperative Information Agents VI*, LNCS Vol. 2446, Springer, Berlin.

Odell, J., Nodine, M. and Levy, R. (2005) 'A meta-model for agents, roles, and groups', in Odell, J., Giorgini, P. and Müller, J. (Eds.): *Agent-Oriented Software Engineering V*, LNCS Vol. 3382, Springer, Berlin.

Odell, J., Parunak, H. and Bauer, B. (2000) 'Representing agent interaction protocols in UML', *Proc. of the 1st International Workshop on Agent-Oriented Software Engineering (AOSE 2000)*, pp.121–140.

Odell, J., Parunak, H. and Bauer, B. (2001) 'Extending UML for agents', in Wagner, G., Lesperance, Y. and Yu, E. (Eds.): *Proc. of the Agent-Oriented Information Systems Workshop (AOIS)*, Austin, 2000, pp.3–17.

OMG (2008) 'Software and systems process engineering meta-model specification v2.0', Object Management Group, available at http://www.omg.org/docs/formal/08-04-01.pdf (accessed on 30 March 2010).

Oyenan, W., DeLoach, S.A. and Singh, G. (2009) 'A service-oriented approach for integrating multiagent system designs', *Proceedings of the Proc. of 8th Intl. Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, 10–15 May 2009, Budapest, Hungary.

Padgham, L. and Winikoff, M. (2002) 'Prometheus: a methodology for developing intelligent agents', *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems*, 15–19 July 2002, Bologna, Italy, ACM, New York.

Poutakidis, D., Winikoff, M., Padgham, L. and Zhang, Z. (2009) 'Debugging and testing of multi-agent systems using design artefacts', *Multi-Agent Programming*, pp.215–258, Springer, Berlin.

Pressman, R. (2010) *Software Engineering: A Practitioner's Approach*, 7th ed., McGraw-Hill, Boston.

Prieto-Diaz, R. and Arango, G. (1991) *Domain Analysis and Software Systems Modeling*, IEEE Computer Society Press.

Robby, DeLoach, S.A. and Kolesnikov, V. (2006) 'Using design metrics for predicting system flexibility', in Baresi, L. and Heckel, R. (Eds.): *Fundamental Approaches to Software Engineering: 9th International Conference, FASE 2006, Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2006, 27–28 March 2006, Vienna, Austria*, LNCS Vol. 3922, pp.184–198, Springer, Berlin.

Royce, W. (1970) 'Managing the development of large software systems', *Proceedings of IEEE WESCON*, pp.1–9.

Royce, W. (1998) *Software Project Management: A Unified Framework*, Addison-Wesley, Reading, MA.

Rumbaugh, J., Jacobson, I. and Booch, G. (2004) *Unified Modeling Language Reference Manual*, 2nd ed., Pearson Higher Education.

Russell, S.J. and Norvig, P. (2003) *Artificial Intelligence: A Modern Approach*, 2nd ed., Prentice-Hall, Upper Saddle River, NJ.

Seidita, V., Cossentino, M., and Gaglio, S. (2006) 'A repository of fragments for agent systems design', *Proceedings of the 7th Workshop from Objects to Agents (WOA 2006)*, 26–27 September 2006, Catania, Italy.

Shoham, Y. and Tennenholtz, M. (1995) 'On social laws for artificial agent societies: off-line design', *Artificial Intelligence*, Vol. 73, Nos. 1–2, pp.231–252.

Tran, Q-N.N. and Low, G.C. (2005) 'Comparison of ten agent-oriented methodologies', in Henderson-Sellers, B. and Giorgini, P. (Eds.): *Agent-Oriented Methodologies*, pp.341–367, Idea Group, Hershey, PA.

van Lamsweerde, A. and Letier, E. (2000) 'Handling obstacles in goal-oriented requirements engineering', *IEEE Trans. on Software Engineering*, Vol. 26, No. 10, pp.978–1005.

van Lamsweerde, A., Darimont, R. and Letier, E. (1998) 'Managing conflicts in goal-driven requirements engineering', *IEEE Transactions on Software Engineering*, Vol. 24, No. 11, pp.908–926.

van Riemsdijk, M.B., Dastani, M. and Winikoff, M. (2008) 'Goals in agent systems: a unifying framework', *Proceedings of the 7th international Joint Conference on Autonomous Agents and Multiagent Systems – Volume 2, International Conference on Autonomous Agents*, International Foundation for Autonomous Agents and Multiagent Systems, pp.713–720.

Winikoff, M. (2009) 'Future directions for agent-based software engineering', *International Journal of Agent-Oriented Software Engineering*, Vol. 3, No. 4, pp.402–410.

Notes

1 Technically, SPEM 2.0 defines task definitions, role definitions, and work product definitions as method content with task uses, role uses, and work product uses being instances of those definitions in actual methods. This paper refers to both forms as tasks, roles or work products.