The ASEME methodology

Nikolaos I. Spanoudakis*

Applied Mathematics and Computers Laboratory,
School of Production Engineering and Management,
Technical University of Crete,
University Campus,
73100, Chania, Greece
Email: nikos@amcl.tuc.gr
*Corresponding author

Pavlos Moraitis

Laboratory of Informatics Paris Descartes (LIPADE),
University of Paris,
45 rue des Saints-Peres, 75006,
Paris, France
Email: pavlos.moraitis@u-paris.fr
and
Argument Theory,
37 rue de Lyon, 75012,
Paris, France
Email: pavlos@argument-theory.com

Abstract: In this paper, we present a complete view of an agent-oriented software engineering methodology called *agent systems engineering methodology* ($ASEME$). Several parts of the methodology concerning different aspects of the whole development process have already been published in the past in several papers. However, our goal in this paper is to provide a global view on the methodology by providing information about the agent (and multi-agent systems) development process along with recent works concerning the tools that we have developed in order to facilitate the use of $ASEME$ by agent systems developers. We also provide some information on the different practical applications that we have developed using $ASEME$ and which prove that $ASEME$ is very well suited for the development of real world applications.

Keywords: software engineering; agent-oriented software engineering; AOSE; intelligent agents; multi-agent systems; methodologies; software process; model-driven engineering.

Reference to this paper should be made as follows: Spanoudakis, N.I. and Moraitis, P. (2022) 'The ASEME methodology', *Int. J. Agent-Oriented Software Engineering*, Vol. 7, No. 2, pp.79–107.

Biographical notes: Nikolaos I. Spanoudakis is a researcher-teaching assistant at the School of Production Engineering and Management of the Technical University of Crete and member of the Applied Mathematics and Computers Laboratory. He holds a PhD in Computer Science from the University of Paris (France), an MSc in Organisation and Management from the Technical University of Crete, and a Diploma in Computer Engineering and Informatics from the University of Patras. His research interests are mainly in the areas of intelligent systems, autonomous agents and multi-agent systems, agent-oriented software engineering, and applications of argumentation. He is a senior member of the Institute of Electrical and Electronics Engineers (IEEE), and a member of the Association for Computing Machinery (ACM). He has gained experience working for more than 12 community or industry funded research projects with the roles of researcher, systems designer and developer, technical and project manager, and scientific coordinator.

Pavlos Moraitis is a Professor of Computer Science at the University of Paris, France, a member of the Laboratory of Informatics Paris Descartes (LIPADE) and the Head of the Distributed Artificial Intelligence (DAI) research group. He is also the CEO and co-Founder of the AI Startup Argument Theory. His research activity concerns both theoretical and applied research in the field of artificial intelligence and more particularly in the area of intelligent agents and multi-agent systems. His main research interests include: computational argumentation, automated negotiation, automated decision making, multi-agent planning and agent-oriented software engineering.

1 Introduction

Agent-oriented software engineering (AOSE) is a research domain concerned with defining metaphors, concepts and methods inspired by the multi-agent systems domain for agent-based software development. Agents are the descendants of objects. They are proactive (have goals and act to achieve them), reactive (respond to events occurring in their environment), social (are acquainted with other similar software and can cooperate-compete with it), autonomous (do not need human intervention to act), and intelligent computational entities, i.e., they may perform such tasks that, when performed by humans, we consider as evidence of a certain intelligence (see e.g., Wooldridge, 2009). The multi-agent systems research area emerged mainly from the artificial intelligence (AI) domain and one of the goals of the AOSE community is to bring agent technology to the mainstream software engineering community.

The agent systems engineering methodology ($ASEME$) is an AOSE methodology for developing agent-based systems. Its origin lies in the Gaia2JADE process (Moraitis and Spanoudakis, 2006) for implementing Gaia models (Wooldridge et al., 2000) using the JADE agent platform (Bellifemine et al., 2007). It emerged as an evolution of the Gaia2JADE process influenced by the requirements analysis phase of Tropos (Bresciani et al., 2004) and the work of Moore (2000) on conversation policies (Spanoudakis, 2009). Existing papers in the literature have focused on specific aspects of the methodology (Spanoudakis and Moraitis, 2008a, 2008b, 2009b, 2010, 2011).

In this paper we present a global view of ASEME, including an introduction to the tools that support the software development process and evaluation. The $ASEME$ process follows the modern model driven engineering (MDE) style (Beydeda et al., 2005), thus, the models of each phase are produced by applying transformation rules to the models of the previous phase. Each phase adds more detail and becomes more formal, gradually leading to implementation.

This paper's originality is the presentation for the first time of the complete ASEME process (in Sections 2 and 3). The complete ASEME process includes the agent society, in contrast to Spanoudakis and Moraitis (2011) that presented the process for single agent development without defining interaction protocols, as well as the description of the ASEME integrated development environment (IDE) and its dashboard. The other previous papers show partial aspects of the whole process, isolated technical tools, or, case studies of real-world systems that illustrate the use of particular models but not the development process, which is the main issue in the current paper.

Then, this paper shows how the previously developed tools can fit together to assist the whole development process (in Section 4). Actually, this paper focuses on the development process with ASEME and not on the models that it uses, which have indeed been covered by the previous papers (Spanoudakis and Moraitis, 2008a, 2008b, 2009b, 2010, 2011; Spanoudakis et al., 2018). This paper aims to aid developers of MAS that want to use the ASEME IDE and its dashboard without needing to have any particular knowledge of the models behind.

Finally, Section 5, evaluation, is also original. It includes a list of references of independent researchers that have used ASEME models and shows the potential application fields and case studies that can help an interested AOSE practitioner. This is important as it shows that people other than us have used it for developing systems.

In the area of AOSE a number of development methodologies has been proposed during the last 20 years (e.g., Bresciani et al., 2004; Cossentino, 2005; DeLoach and Garcia-Ojeda, 2010; Garijo et al., 2005; Henderson-Sellers and Giorgini, 2005; Iglesias and Garijo, 2005; Padgham and Winikoff, 2004; Pavon et al., 2005; Picard and Gleizes, 2004; Wooldridge et al., 2000). Some methodologies, e.g., Tropos (Perini and Susi, 2006), have proposed MDE processes for some phases of the development process.

MDE is based on metamodelling (Gascueña et al., 2012), and the Ingenias metamodel (Pavon et al., 2005), for instance, is one of the richest among AOSE methodologies containing more than 300 concepts. García-Magariño et al. (2009) presented an algorithm for defining model transformations by-example. Their approach allows the engineers to define themselves the transformations that they want to apply to Ingenias models. Another interesting work is presented by Jayatilleke et al. (2005), where the authors propose a component-based approach to designing belief-desire-intentions (BDI) architectures (Georgeff et al., 1999). Their work focuses in defining the metamodel for BDI-relevant entities as goals, events, triggers, plans, actions, beliefs, and, finally, agents and then in defining the transformation to code for the JACK agent platform (Winikoff et al., 2005). Hahn et al. (2008) defined a metamodel (PIM4Agents) that can be used to model MAS in the platform independent model (PIM) level of the model-driven architecture (MDA). The added value of their work is that PIM4Agents instances can be instantiated with both the JADE (Bellifemine et al., 2007) and JACK agent platforms.

$ASEME$ offers some unique characteristics regarding the used MDE approach. It covers all the classic software development phases (from requirements to implementation) and the transition of one phase to another is done through model transformations. Thus, the analysts/engineers and developers just enrich the models of each phase with information, gradually leading to implementation. However, its main advantage over others is that it allows non specialists in the multi-agent systems domain to take advantage of the added value of agent technology by using familiar modelling languages and having most of the MAS part of the required code automatically generated.

Moreover, for design phase models, $ASEME$ employs the language of statecharts (Harel and Naamad, 1996), quite popular in AOSE for modelling interaction protocols (Moore, 2000; Paurobally et al., 2004). DeLoach et al. (2001) used statecharts for coordinating the agent roles, introducing the term *intra-agent control* (IAC), while other researchers used them for planning (Nwana et al., 1999; DeLoach et al., 2001; Murray, 2004). In $ASEME$ both the *inter-* and *intra-agent control* models are defined using the statechart formalism that allows for seamless integration of agent capabilities and interaction protocols.

2 *ASEME* method concepts

$ASEME$ is a methodology for developing autonomous agents and multi-agent systems. It uses the agent modelling language ($AMOLA$) for modelling agent-based systems. The latter provides the syntax and semantics for creating models of agents and multi-agent systems covering the analysis and design phases of the $ASEME$ software development process.

$AMOLA$ supports a modular agent design approach and introduces the concepts of *IAC* and inter-agent control (EAC). The first defines the agent's lifecycle by coordinating the different modules that implement his capabilities, while the latter defines the protocols that govern the coordination of the society of the agents composing a multi-agent system. The modelling of the IAC and EAC is based on statecharts (see e.g., Harel and Naamad, 1996). The concept of *capability* is defined as the ability to fulfill specific tasks. Capabilities are decomposed to simple *activities*. The capabilities correspond to modules that are integrated using the IAC concept to define an agent. The concepts of capability and functionality are distinct, in contrast to other works where they refer to the same thing but at different stages of development, e.g., in Prometheus (Padgham and Winikoff, 2004).

$AMOLA$ is compatible with the object management group's (OMG) MDA paradigm (Kleppe et al., 2003). According to that, the model-driven development process includes the definition of three important models:

- The computation independent model (CIM). It describes what the system should accomplish, hiding the information technology part. The CIM of $AMOLA$ is the system actors and goals (SAG) model.
- The PIM. The PIM is a technical model describing the system's functionality, hiding the implementation details. It is a system design. The PIM of $AMOLA$ is the IAC model.

- The platform specific model (PSM) defines an implementation of the PIM in a specific platform.

The *ASEME* development process is presented through a *software process* which is defined as a series of *phases* that produce *work products*. The software development phases of *ASEME* are presented in Figure 1 using the extended SPEM 2.0 language for representing agent oriented methodologies (Seidita et al., 2009).

Figure 1 ASEME process overview (see online version for colours)

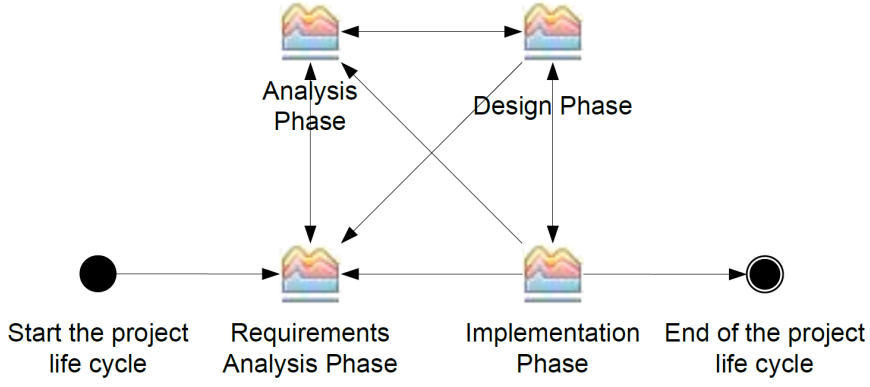


Figure 2 ASEME phases and their products per level of abstraction

Development Phase	Society Level (1st level of abstraction)	Agent Level (2nd level of abstraction)	Capability Level (3rd level of abstraction)
Requirements Analysis	Actors System Actors Goals (SAG)	Goals SAG	Requirements SAG
Analysis	Roles and Protocols System Use Cases (SUC), Agent Interaction Protocols (AIP)	Capabilities SUC, System Roles Model (SRM)	Functionalities SRM
Design	Society Control intEr-Agent Control (EAC), Ontology	Agent Control Intra-Agent Control (IAC)	Components IAC
Implementation	Platform management code Platform Specific Model (PSM)	Agent code PSM	Capabilities code PSM

The *ASEME* process is iterative, allowing for incremental development. In *ASEME*, the *SAG* model, the systems roles model (*SRM*), the *IAC* model and a *PSM* (presented below) are the main models outputted by the requirements analysis, analysis, design and implementation phases respectively. Each of these models is produced by transforming the previous phase model. The forward arrows in Figure 1 (i.e., those that point clockwise from requirements analysis to implementation) imply the use of a transformation process, while the backward arrows imply that the modeler returns to

the models that he edited in the targeted phase. The project may follow a number of iterations before finishing.

Here we would like to note that the fact that several development phases such as testing, checkout, maintenance, etc. (Gomez-Sanz et al., 2011) are not shown in Figure 1. They are assumed to be carried out using classic software engineering techniques, i.e., $ASEME$ does not define its own. They can be assumed to tail the implementation phase.

Three levels of abstraction are defined for each phase. The first is the *societal level*. There, the whole multi-agent system behaviour is modelled. Then, the second level, or the *agent level*, zooms in each part of the society, i.e., the agent. Finally, the details that compose each of the agent's parts are defined in the third level, *the capability level*. $ASEME$ is mainly concerned with the first two abstraction levels assuming that development in the capability level can be achieved using classical (or even technology-specific) software engineering techniques. In Figure 2, the $ASEME$ phases, the different levels of abstraction and the models related to each one of them are presented.

All the depicted models are defined by $AMOLA$, except ontology, which can be defined using any relevant formalism, e.g., the UML class diagram. In the following section each of these phases will be enriched with a process definition.

3 The *ASEME* process

An example on how to develop a meetings management system is used throughout this section for the $ASEME$ process demonstration. This example (the meetings management system) has been widely used in the past for demonstrating the use of AOSE methodologies, e.g., for the Prometheus and MAS-CommonKADS methodologies (Henderson-Sellers and Giorgini, 2005). This system's requirements are, in brief, to support the meetings arrangement process. The user needs to be assisted in managing his meetings by a personal assistant. The latter manages the user's schedule and services the user. The meetings organisation process is managed by the secretariat to which the users submit their requests to schedule a new meeting or change the date of an existing one. The secretariat contacts the users' assistants whenever she needs to negotiate a meeting date.

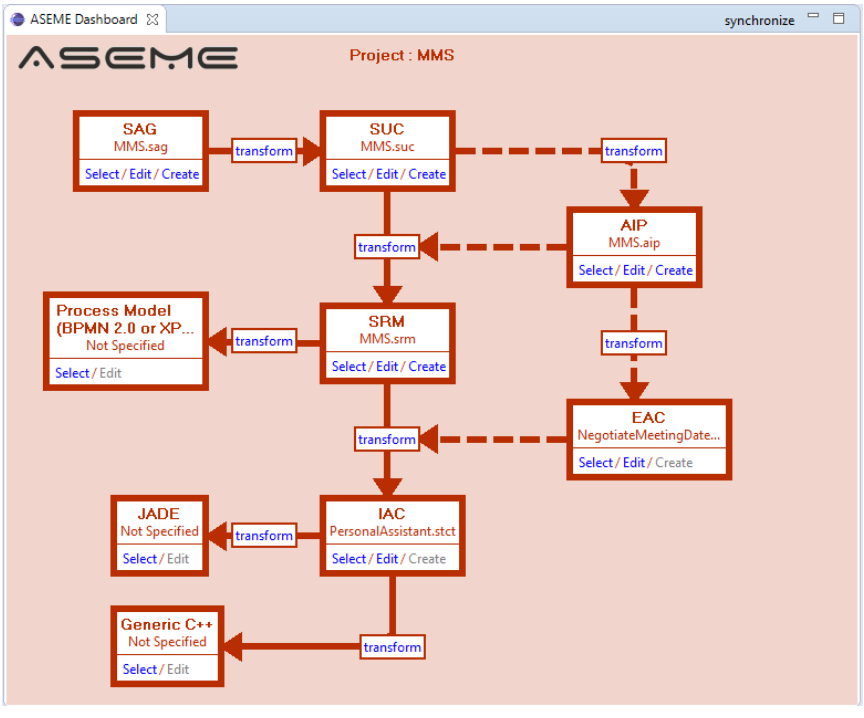
The $ASEME$ process is facilitated by the $ASEME\ dashboard\ tool$, which guides the modeler from capturing requirements to implementation. It follows the style of similar graphical user interfaces (GUI), such as the GMF dashboard of the Eclipse Modeling Tools Project (Taentzer et al., 2008). It is depicted in Figure 3, where the solid lines show the mandatory parts of the software development process, which is usually followed for single (autonomous) agent development. The dashed lines show the optional parts, related to agents interaction modelling, which must also be used when developing multi-agent systems.

Each box in Figure 3 is titled with the model type it represents, e.g., *SAG* for the top left box. Under the model type the user can see the model instance that has been edited or generated in the specific project, e.g., *MMS.sag*. The user can *select* a new model (by browsing the project folder), *create* a new model or *edit* the depicted model by clicking the relevant link at the bottom of the box. By pressing the 'transform' button between

the boxes the relevant transformation tool is invoked taking as input the model in the arrow's source box and generating one model in the target box.

Note that the modeler can start at whichever step he/she likes depending on the familiarity with $ASEME$, the problem domain and the complexity. Someone might start with the SUC model, another with the SRM.

Figure 3 The $ASEME$ dashboard (see online version for colours)



3.1 Requirements analysis phase

In the requirements analysis phase and in the first level of abstraction, the actors and their goals that depend on other actors are defined; in the second level, the individual goals of each actor are identified, and, in the third level, specific requirements, functional and non-functional, are associated to each one of these goals. The output of the requirements analysis phase is the SAG model, containing the actors and their goals which have been associated with requirements. All these activities are usually performed by a business consultant (a representative of the organisation that will develop the software) together with a firm representative (who represents the client).

3.1.1 SAG model

The $AMOLA$ model for the requirements analysis phase is the SAG model, a graph involving actors and goals, inspired by the Tropos actor diagram (Bresciani et al., 2004).

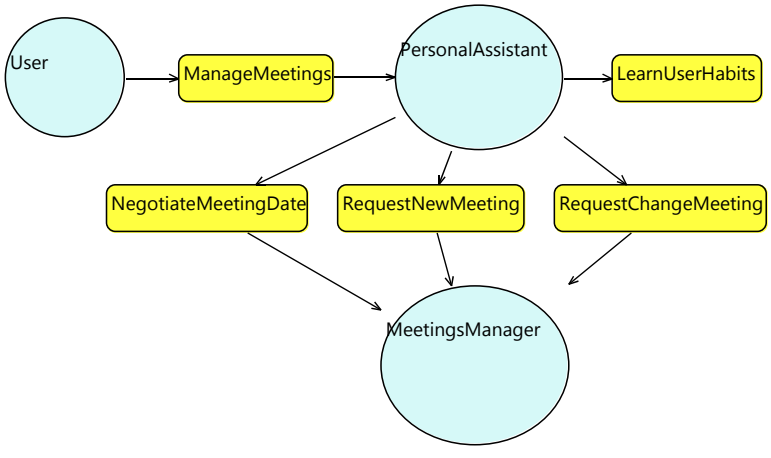
A goal of one actor (owner of the goal) may be dependent for its realisation to another actor (collaborator). The owner actor depends on the collaborator(s) to achieve the goal.

Graphically, actors are represented as circles and goals as rounded rectangles. Dependencies are navigable from the owner to the goal and from the goal to the collaborator(s). The goals are then related to functional and non-functional requirements in plain text.

An entity can qualify as an actor if it represents a real world entity (e.g., a 'broker', the 'director of the department', a 'shop', etc). Some of these actors, as we will show later, will emerge as agents during the system analysis phase. Summarising, the SAG model consists of *goals* and *actors*.

Regarding the running example for the meetings management system, the actors involved are the user and the assistant (or secretary) that helps him to manage his meetings. Moreover, there is the department secretariat role that is represented by the meetings manager actor. The reader can see the SAG model in Figure 4. The goal of the user to manage his meetings is dependent on the personal assistant. In the agent level individual goals are defined; one such for the personal assistant is the adaptation to user needs, named 'learn user habits'. In the capability level the functional and non-functional requirements for each goal are defined in free text. A non-functional requirement for the user's *manage meetings* goal could be to be able to 'be executed on a mobile device'. Another is that it should 'reply to a user request within ten seconds'. The requirements can be edited in the properties field of the selected goal in the graphical model (they are not shown on the graphical model as they would clutter the diagram).

Figure 4 The SAG model (see online version for colours)



3.2 Analysis phase

The first task of the analysis phase is to transform the SAG model to a system use case (SUC) model.

3.2.1 The SUC model

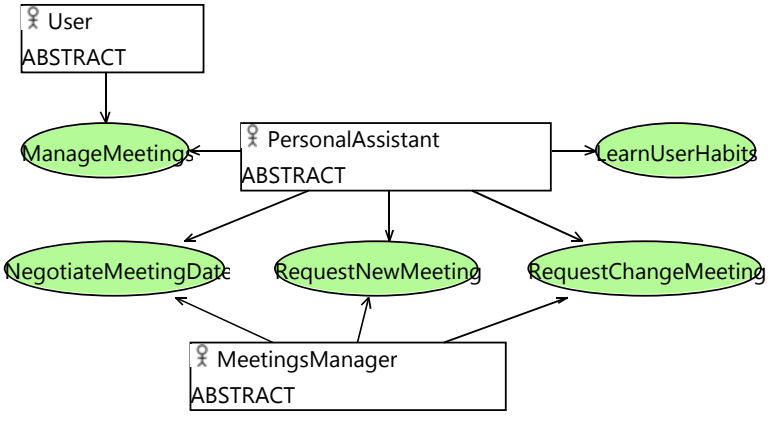
The SUC model is similar to the use case diagram of UML (OMG, 2007). It helps to visualise the system in terms of roles and tasks that they realise. Moreover, it allows decomposing a complex task to simpler ones. It includes system interaction with external entities, be they humans or other systems. No new elements are needed other than those proposed by UML. However, the semantics change. Firstly, the actor 'enters' the system and assumes a role. In the UML use case diagrams the actor is always a user. In the SUC diagrams the actors may be humans, but also *system roles*, indicating an agent role, either within the system or outside it (for existing systems in the environment).

The general use cases can be decomposed to simpler ones using the *include* relationship. General use cases are also referred to as *capabilities*. A use case that connects two or more (agent) roles implies the definition of a special capability type: the participation of the agent in an *interaction protocol*. A use case that connects a human and an artificial agent implies the need for defining a *human-machine interface* (HMI), another agent capability. A use case can include a second one showing that its successful completion requires that the second also takes place.

The SUC model is initialised automatically by the SAG model. The *SAG2SUC* transformation maps concepts from the SAG model to those of the SUC model. Figure 5 shows the produced SUC model for our MMS example. The actors are transformed to roles and the goals to use cases. The transformation is straightforward and someone might wonder why do we need the SAG diagram. In our view, use cases are much more formal than SAG. For experienced engineers the use case text is not just free text like the requirements field of SAG goal. SAG corresponds to the non-technical CIM level of MDA. Moreover, the use case that is derived by a goal is connected to the roles which are derived from actors that were related to the goal either as owners or as collaborators.

Note that the relationships between the roles and use cases are always directed from the role to the use case (as in the UML use case diagrams) as when it comes to interaction both roles will have to do some tasks regardless of who depends on whom. All roles are initially of type 'abstract'.

Figure 5 The initial SUC model (see online version for colours)

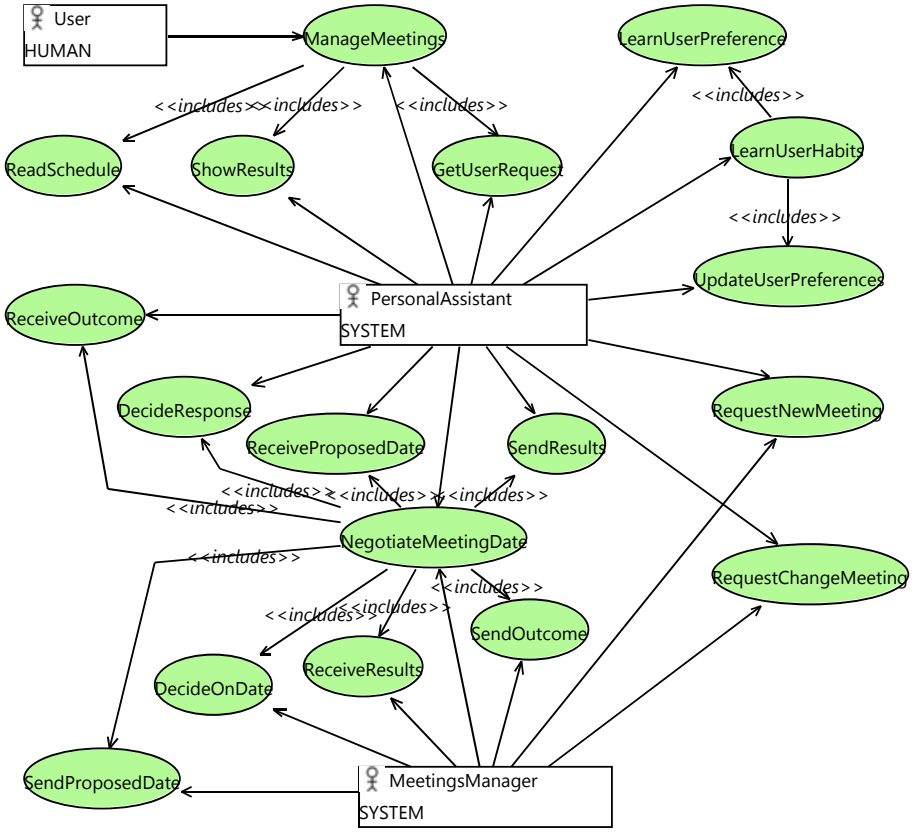


Note: Transformed from Figure 4.

The analysts can refine the use cases using the *include* relationship and by editing the *specified_by* property to transform the goal requirements to task definitions (e.g., ordering the actions done by the actor, defining pre-conditions, post-conditions and alternative flows). All the use cases connecting two or more system roles concern the society level, while the use cases that have only one role participant concern the agent level. In the society level, the analysts can choose to create more roles and define interactions between them. In the agent level, the analysts will eventually decompose the general use cases to more elementary ones.

Figure 6 shows the *SUC* model where the *NegotiateMeetingDate* use case has been refined and includes several use cases corresponding to the tasks that need to be achieved by the *PersonalAssistant* and *MeetingsManager* roles. The latter are defined to be 'system' roles, while the user is a 'human' role.

Figure 6 The refined *SUC* diagram (see online version for colours)



For example, the *NegotiateMeetingDate* use case for the personal assistant has been decomposed to more elementary ones, i.e.,

a to receive a proposed date (*ReceiveProposedDate*)

b to decide the user's response (*DecideResponse*)

c to send the outcome of the decision (*SendResults*)

d to receive the outcome of the negotiation process (*ReceiveOutcome*).

3.2.2 The agent interaction protocols model

Protocols (in the society level) originate from use cases that connect two or more roles. The agent interaction protocol (AIP) model is composed of the following elements:

- Protocol
  - a name (a String type property for storing the name of the protocol)
  - b participants (a list of two or more references to participant instances).
- Participant
  - a name (a String type property for storing the name of the participant)
  - b engaging_rules (a String type property for storing the specifications of the prerequisites for the participant to enter the protocol in free text format)
  - c outcomes (a String type property for storing the possible outcomes of the protocol in free text format)
  - d liveness (the process that the participant would follow for achieving the objective of the interaction in the form of a liveness formula).

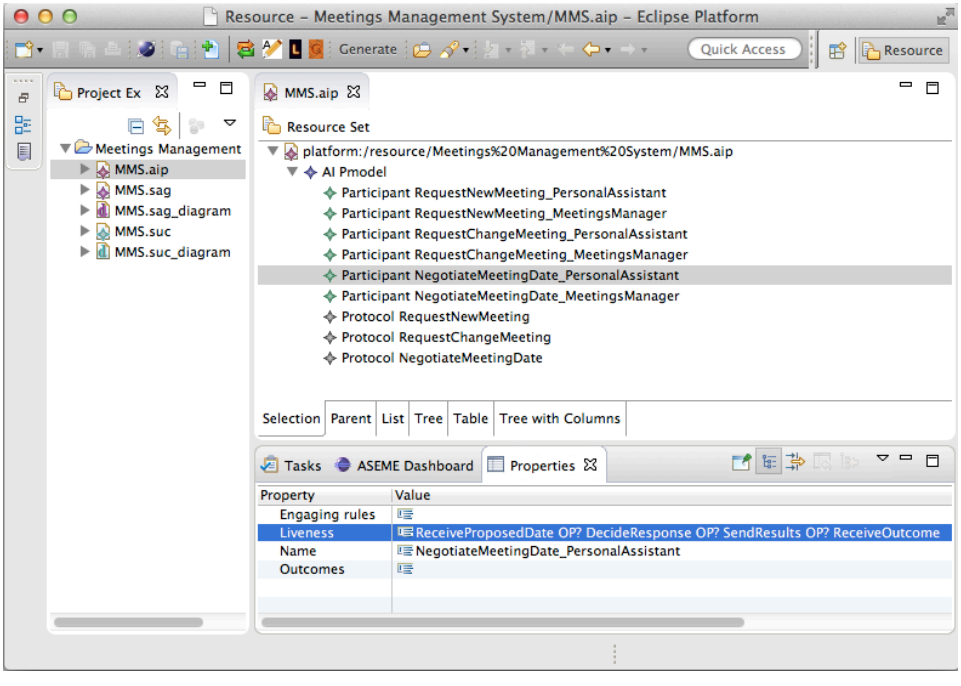
The *liveness formula* is a process model that describes the dynamic behaviour of the role. It connects all the role's activities using the Gaia operators (Wooldridge et al., 2000). Briefly, $A.B$ means that activity B is executed after activity A, $A\sim$ means that activity A is executed forever (when it finishes it restarts), $A|B$ means that either activity A or activity B is executed and $A||B$ means activity A is executed in parallel with activity B. Additionally, $[A]$ means that activity A is optional (it may be executed), $A*$ means that activity A will be executed zero or more times and $A+$ means that activity A will be executed one or more times. Note that we have replaced the original omega (ω) operator of Gaia with the tilde ($\sim$) as it is more practical and quickly available in most keyboards.

The AIP model is automatically initialised from the SUC model. One protocol is created for every use case that has more than one role participants. The $SUC2AIP$ transformation tool initialises the process part of each protocol participant adding all the included use cases connected with the 'OP?' symbol. This is a general feature that characterises the $ASEME$ development process aiming to ensure that the modeler will not forget or lose part of the information he has already supplied in a previous model. Then, the modeler has to put the use cases in the right order and connect them with the appropriate Gaia operators.

In Figure 7, the reader can see the MMS.aip model introduced in the workspace (left hand side) after the execution of the $SUC2AIP$ program (which is invoked transparently to the user when he clicks the transform button connecting the two models in the $ASEME$ dashboard). The *PersonalAssistant* participant of the *NegotiateMeetingDate* protocol is selected and at the bottom the modeler edits its properties.

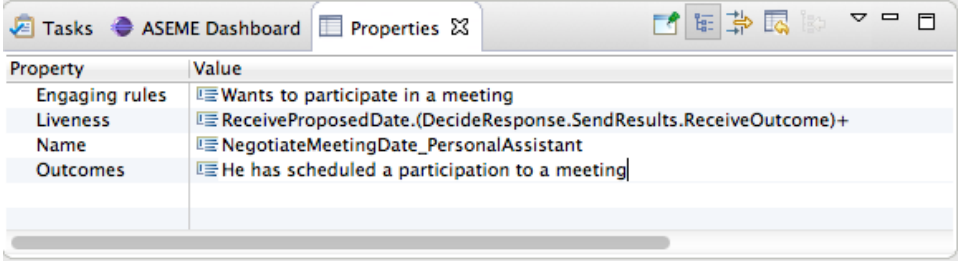
Figure 8 shows the properties again after they have been edited by the modeler. Now the reader can see that the Liveness property of the personal assistant role in the *NegotiateMeetingDate* protocol contains the previously identified elementary use cases – transformed to activities and the added value in this model is that they have been used for defining the process that the personal assistant will use in the protocol. Thus, the role first receives the proposed date (*ReceiveProposedDate*) and then the following activities take place one before the other one or more times (notice the + Gaia operator): decide the user's response (*DecideResponse*), then send it (*SendResults*) and then receive the outcome of the negotiation round (*ReceiveOutcome*).

Figure 7 The automatically generated AIP model (see online version for colours)



Notes: Screenshot taken from a computer with MacOS, Eclipse Modeling Tools Luna R2 and $ASEME$ v.2.1.

Figure 8 The refined *PersonalAssistant* participant of the *NegotiateMeetingDate* protocol (see online version for colours)



3.2.3 The SRM

The SRM is mainly inspired by the Gaia roles model (Wooldridge et al., 2000). A role model is defined for each SUC system role and contains the following elements:

- Activity
  a name (a String type property for storing the name of the activity)
  b functionality (the functionality related to this activity).
- Capability
  a name (a String type property for storing the name of the capability)
  b activities (a list of zero or more references to activity instances).
- Role
  a name (a String type property for storing the name of the role)
  b capabilities (a list of zero or more references to capability instances)
  c activities (a list of zero or more references to activity instances)
  d liveness (the process that the role follows in the form of liveness formulas).

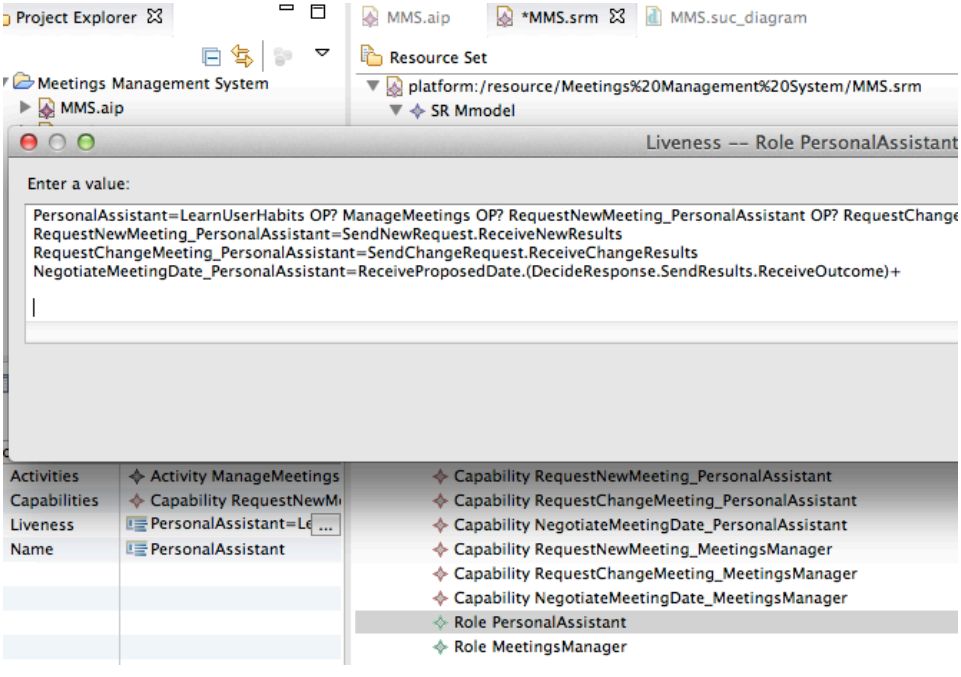
Therefore, a role aggregates capabilities and capabilities aggregate activities (these terms have been discussed in Section 2). In the liveness property of the role, its name appears in the left hand side of the first formula (root formula). Activities or capabilities can be added on the right hand side connected with Gaia operators. A capability must be decomposed to activities or more capabilities in a following formula.

The SUC model is transformed to the SRM. An SRM role is created for each SUC role with type *system*. The use cases that include others are inserted as capabilities, while the included ones as activities. If an AIP model is available then the capabilities related to protocols are expressed as such and the liveness formulas of the role's participation in one or more protocols are imported to the role's liveness model. See, e.g., in Figure 9 the fourth formula, which is the same with the personal assistant role's liveness in the *NegotiateMeetingDate* protocol in the AIP model (presented in Figure 8). In Figure 9, notice that all the use cases connected with the personal assistant role, except those that are included by them, participate on the right hand side of the first formula. Behind the liveness formula (in the figure) is the SRM model and the user has selected the *PersonalAssistant* role. On the left the properties of the role are visible (just beneath the liveness window).

After the $SUC2SRM$ transformation, the analysts refine the liveness formulas, connecting capabilities and activities with the appropriate Gaia operators (see the refined SRM model in Figure 10). For example, at the first formula, the right hand side expression means that the role executes the *ManageMeetings* capability, followed by the *LearnUserHabits* capability. This is repeated forever (notice the $\sim$ operator), in parallel (notice the || operator) with the personal assistant's capability to participate in the *NegotiateMeetingDate* protocol (itself repeated forever). The *ManageMeetings* capability

process is defined in the second formula, the *LearnUserHabits* in the third formula and the *NegotiateMeetingDate_PersonalAssistant* in the sixth formula.

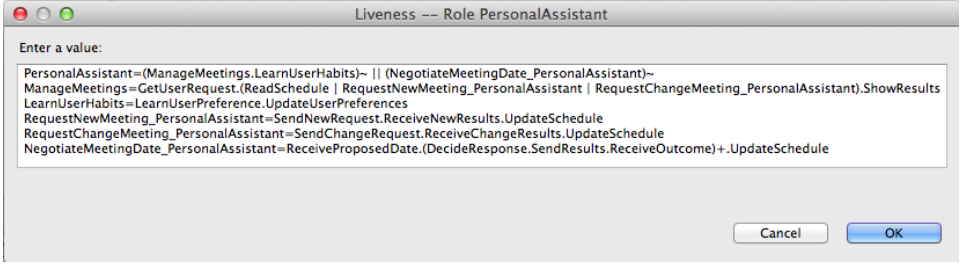
Figure 9 The generated *PersonalAssistant* role in the SRM model (see online version for colours)



Notes: The edit dialog shows the liveness property of the role.

The analysts must take care not to tamper with the protocol parts in the liveness formula, as the agent must comply with a protocol specification. However, the analysts are free to determine how to achieve the tasks specified by the protocol, either as activities, or as capabilities. In the latter case, the analysts can define a new formula whose left hand side is the protocol activity that needs to be expanded as a new capability.

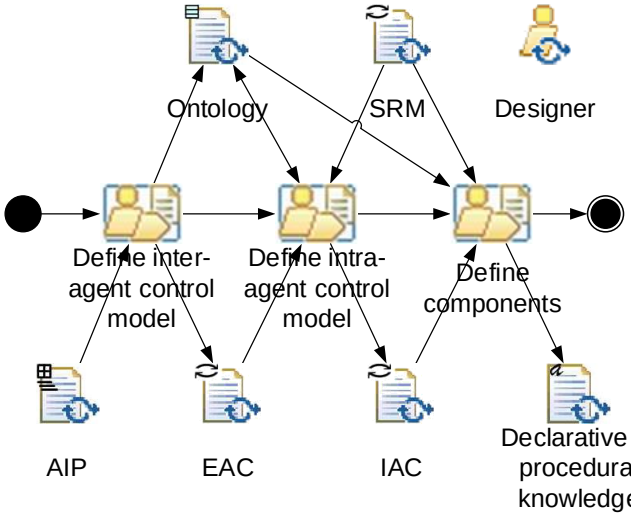
Figure 10 The refined liveness formula of the *PersonalAssistant* role in the SRM (see online version for colours)



3.3 Design phase

The $ASEME$ design phase process is presented in Figure 11. The three activities reflect the three different levels of abstraction in this software development phase. In the society level we have the definition of the *EAC* model, in the agent level the definition of the *intra-agent control* model and in the capability level the definition of the different components that will be used by the agent. In this diagram the reader can also see the input and output models of each activity (the arrow direction from the resource to the activity shows that it is an input, while the reverse indicates that it is an output).

Figure 11 The $ASEME$ design phase (see online version for colours)



The agents communicate using interaction protocols that are described by the *EAC*, which defines the participating roles and their responsibilities in the form of tasks. The agents implement the roles that they can assume through their capabilities. The capabilities are the modules that are integrated using the IAC concept. The first activity ('define EAC model') consists of four tasks and produces the EAC model, the set of the performatives of the inter-agent messages and the ontology that will be used.

3.3.1 The EAC model

The EAC is defined as a statechart (Harel and Naamad, 1996). It is initialised by transforming the AIP of the analysis phase to statecharts. Statecharts are used for modelling systems. They are based on an activity-chart that is a hierarchical data-flow diagram, where the functional capabilities of the system are captured by activities and the data elements and signals that can flow between them. The behavioural aspects of these activities (what activity, when and under what conditions it will be active) are specified in statecharts. The fact that the statechart can capture together the functional and behavioural aspects of a system is its greatest advantage (Harel and Naamad, 1996). This is not true for a single UML model as a number of different models need to be combined for a complete description of a system (e.g., a class diagram together

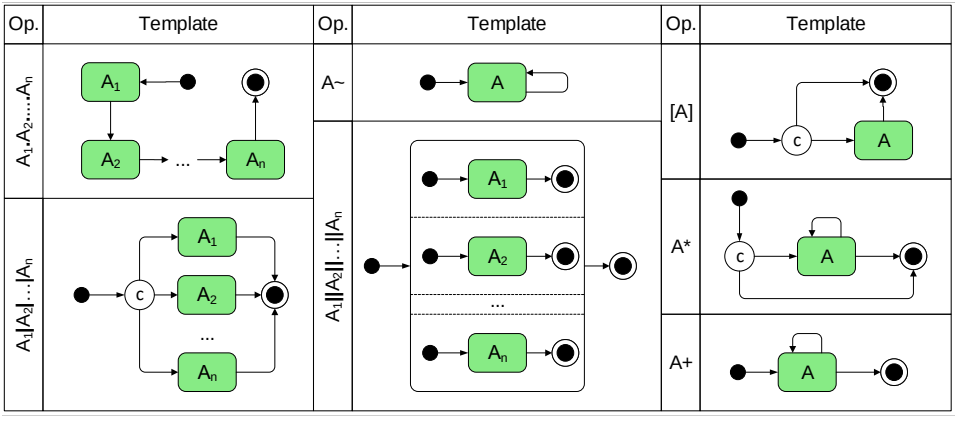
with an activity diagram). Thus, statecharts are ideal for defining systems in a platform independent manner. We use statecharts in different levels of abstraction, firstly in the agent society level, in order to model the interactions between its agents, and, secondly, in the agent level, in order to model the interactions between its components (or capabilities). The statechart, therefore, implements the EAC model in the society level of abstraction, and the IAC model in the agent level of abstraction.

Multiple concurrently active statecharts are considered to be orthogonal components at the highest level of a single statechart. If one of the statecharts becomes non-active (e.g. when the activity it controls is stopped) the other statecharts continue to be active and that statechart enters an idle state until it is restarted. Each transition from one state (source) to another (target) is labelled by an expression, whose general syntax is $e[c]/a$, where e is the event that triggers the transition; c is a condition that must be true in order for the transition to be taken when e occurs; and a is an action that takes place when the transition is taken. All elements of the transition expression are optional. Transitions are usually triggered by events. Such events can be a sent or received (or perceived, in general) inter-agent message, a timeout, and, the completion of the executing state activity.

Having defined the statechart as it is used in $AMOLA$ it is now possible to proceed to the definition of the EAC model. The EAC is a statechart that contains an initial (START) state connected to an AND-state named after the protocol, which in turn connects to a final (END) state. The AND-state contains as many OR-states as the protocol roles named after the roles.

The $AIP2EAC$ tool transforms the process part of the AIP model to the EAC model. A state diagram is generated by an initial AND-state named after the protocol. Then, all participating roles define OR sub-states. The right hand side of the liveness formula of each role is transformed to several states within each OR-state by interpreting the Gaia operators in the way described in Figure 12.

Figure 12 Templates of extended Gaia operators for statechart generation (see online version for colours)



The liveness model for the EAC model for a protocol named $protocol\_name$ including n roles is the following:

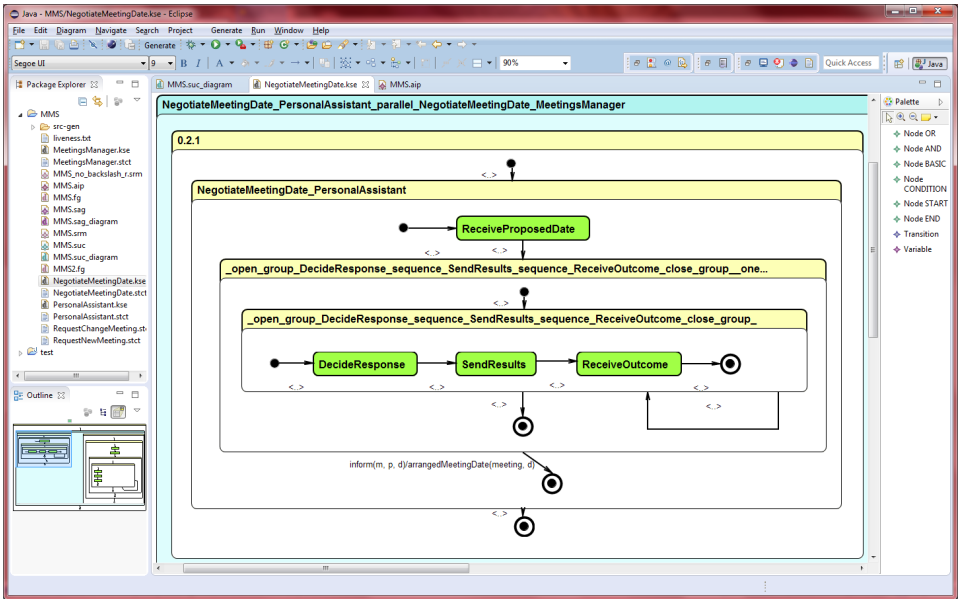
$$protocol\_name = (role\ 1\ process)||(role\ 2\ process)||...||(role\ n\ process)$$

For the case of the meetings management system the liveness formula for the 'negotiate meeting date' protocol is:

$$NegotiateMeetingDate = MeetingsManager||PersonalAssistant$$
$$MeetingsManager = DecideOnDate.SendProposedDate.$$
$$(ReceiveResults.DecideOnDate.SendOutcome) +$$
$$PersonalAssistant = ReceiveProposedDate.$$
$$(DecideResponse.SendResults.ReceiveOutcome)+$$

After applying the transformation algorithm, the statechart depicted in Figure 13 is created. The *ReceiveProposedDate* state is a basic state (drawn as a green-coloured rounded rectangle), while the *NegotiateMeetingDate_PersonalAssistant* state is an OR state (drawn as a yellow-colour-labelled rounded rectangle that contains other states), as in the next formula this capability is further expanded. A node with a circled 'c' represents a condition-state; solid black nodes correspond to start-states and circled black nodes to end-states.

Figure 13 The EAC model as it was produced by the $AIP2EAC$ transformation with one transition expression added (see online version for colours)



Notes: The figure shows the personal assistant role's part of the protocol. Screenshot taken from a computer with Windows 7, Eclipse Modeling Tools Luna R2 and $ASEME$ v.2.1.

Then, the designer defines the message *performatives* set P allowed within the protocol. For our MMS example, $P \in \{accept, propose, reject, inform\}$. The items that the designer must define at the next $ASEME$ task are the data structures used for defining the protocol messages content (also referred to as the ontology) and the timers (i.e., events that are fired after a certain time period has elapsed).

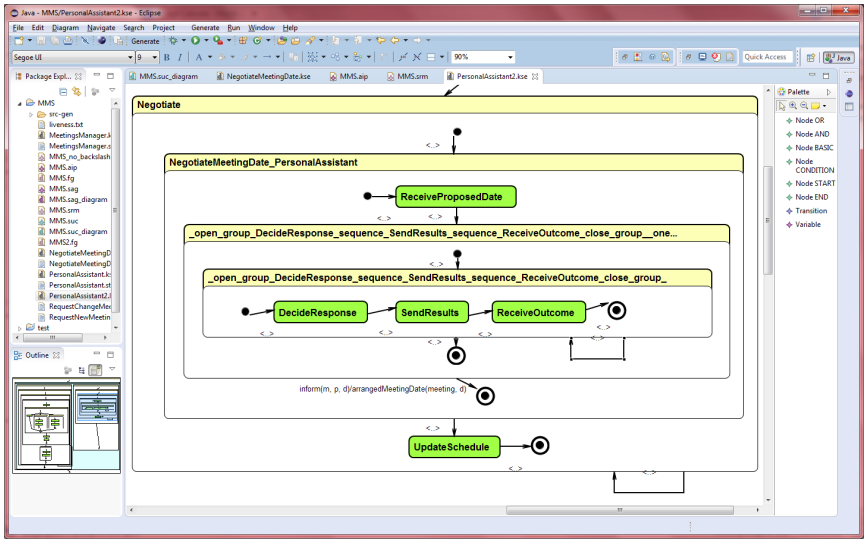
Finally, in the last task of the 'define EAC model' $ASEME$ activity, the transition expressions are defined (see Spanoudakis and Moraitis, 2008a for the details). The preconditions of the AIP become the conditions of a transition from a START state that targets the first state of the protocol for each role. In Figure 13 the modeler has inserted a transition expression using as an event the message $inform(m, p, d)$, where m refers to the meetings manager, p to the personal assistant and d to a date, and then, adding as an action the atom $arrangedMeetingDate(meeting, d)$. This transition expression does not have a condition.

3.3.2 The IAC model

In the agent level, the IAC is created using statecharts in the same way with the EAC model. The difference is that the top level state (root) corresponds to the modelled agent (which is named after the agent type). One IAC is defined for each agent type. The IAC is initialised by transforming the liveness model of the role (SRM) to a state diagram (IAC). This is achieved again by interpreting the Gaia operators in the way described in Figure 12.

Initially, the statechart (IAC) has only one state named after the left-hand side of the first liveness formula of the role model (typically the role's name). Then, this state acquires substates. The latter are constructed by reading the right hand side of the liveness formula from left to right, and substituting the operator found there with the relevant template in Figure 12. If one of the states is further refined in a next formula, then new substates are defined for it in a recursive way. Figure 14 presents the IAC model that is produced by the transformation process ($SRM2IAC$) if its input is the refined SRM shown in Figure 10. The main window of the tool shows a part of the statechart (the whole statechart is outlined on the bottom-left part of the screen), specifically, the one related to the second formula of the SRM (from Figure 10).

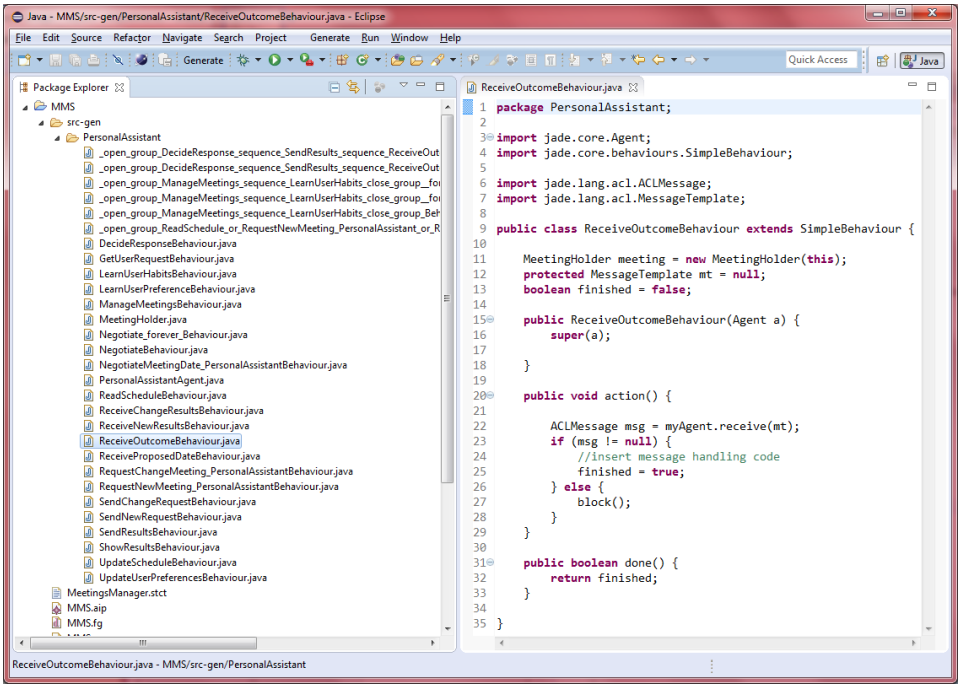
Figure 14 The IAC model produced with the SRM2IAC transformation (see online version for colours)



The transition expressions from the EAC for the part of the statechart containing a protocol (i.e., the part of the statechart produced from the formula whose left hand side is a protocol capability) are imported from the relevant EAC model. In Figure 14 the reader can notice the transition expression [i.e., $inform(m, p, d)/arrangedMeetingDate(meeting, d)$] that we entered earlier in the EAC model, which has automatically been inserted in the IAC model. Finally, the designer enriches the rest of the statechart with transition expressions, updating the ontology, if necessary.

At this point the last things that need to be done are to design the activities that are executed in each state. The input needed is the 'functionality graph' to indicate the technology (e.g., which library to import and which programming language to use), the 'ontology' to show the data structures that will be used by this activity and the 'IAC model' that lists all the activities as states. The output depends on the technology used for each activity and can be based on declarative knowledge (i.e., that considers what should be done, not how it should be done), or procedural knowledge (i.e., that considers how to do something), or both.

Figure 15 The PersonalAssistant Java package automatically generated by the IAC model by the $IAC2JADE$ transformation (see online version for colours)



3.4 Implementation phase

The implementation phase's goal is to transform the PIM to a PSM. This phase can have different instantiations according to the implementation platform. The implementation phase details a transformation process of the PIM to a PSM. The IAC model can be transformed to any language that is supported by a statecharts-based computer-aided

software engineering (CASE) tool. However, it is important to provide a transformation process for an agent development platform as the $ASEME$ process is about agent development.

Herein we provide an overview of the method fragment for the transformation process of the IAC model to agent code using the $JADE$ agent development platform (Spanoudakis and Moraitis, 2010). The JADE platform was selected for demonstrating the capability to transform the IAC model to an agent implementation as is the most popular agent platform and it is an open source software.

Using this process the developer can automatically generate all JADE agent and behaviour classes that will be needed along with the classes representing the IAC model used variables. Moreover, a large part of the needed code is automatically generated, or even the totality of the code, depending on the behaviour type. In Figure 15 the reader can see on the left hand side the Java classes automatically generated when the user hits the transform button from the IAC model to the JADE model. They include the agent class, holder classes for all the variables used in the IAC model, and the JADE Behaviour classes. The latter may need to be connected to the implemented functionalities programs in their action methods. Actually, the control part of the code is created automatically. Indicatively, in a specific project (ASK-IT) the 26% of the code was reported to have been generated automatically (Spanoudakis and Moraitis, 2010). In Figure 15 the *ReceiveOutcomeBehaviour* class is shown on the right hand side. The meetings variable of type *MeetingHolder* is automatically declared as a property of the class. Moreover, the *action* part of the behaviour to receive the message is automatically generated.

4 Tools

A set of tools supporting all the steps of the process discussed in the previous chapter were recently integrated in a development environment along with a number of extensions. All the tools have been developed using the Eclipse Modeling Framework (EMF) (Steinberg et al., 2008; Gascueña et al., 2012) and they are freely available from *github* (http://github.com/nspan/ASEME). The interested reader can be guided to downloading binaries and sources from the $ASEME$ project website (http://aseme.tuc.gr). Specifically, there are graphical editors for the SAG, SUC, AIP, SRM, EAC and IAC models and the model transformation tools $SAG2SUC$, $SUC2SRM$, liveness2statechart (Spanoudakis and Moraitis, 2009b), liveness2BPMN (Spanoudakis et al., 2018), IAC2JADE (Spanoudakis and Moraitis, 2010), IAC2Monas (Paraschos et al., 2012), GGenerator (Papadimitriou et al., 2014), and a CASE tool, KSE (Topalidou-Kyniazopoulou et al., 2013).

All model transformation tools assume that a valid model has been defined in the previous phase. Liveness formulas and transition expressions validation is done through conformance to their relevant BNF (Naur et al., 1963) syntactical rules (Spanoudakis and Moraitis, 2009b), while we have defined validation rules for the statecharts using the GMF tooling (Topalidou-Kyniazopoulou et al., 2013).

Code generation is currently supported for the JADE platform (in the Java programming language), the Monas Robotic platform (and the C++ language), for C++ code connected to any platform through a generic blackboard interface (Papadimitriou et al., 2014), and the business process modelling notation (BPMN) a standard supported

by OMG. The latter (supported by the $Liveness2BPMN$ tool) has been used for system validation and simulation even from the analysis phase (the reader can notice the transformation to BPMN from the SRM model, which is an analysis phase model). Specifically, in the ASK-IT project (Moraitis and Spanoudakis, 2007), the authors used the liveness formulas of SRM to automatically create the process model of a service protocol (Spanoudakis et al., 2018). They showed how a modeler can use the process model to detect flows in the system analysis and design and verify the system's behaviour according to its requirements but also see how it could scale. For example, in the ASK-IT project, there was a requirement that the system should respond to a user request within ten seconds, given that there would be one user request every 30 seconds. Using process simulation tools available in the market the authors validated this requirement and also showed that the system would scale up to service one user request every three seconds without problem, by adding more agent instances of a specific type (Spanoudakis et al., 2018).

The $IAC2Monas$ tool was developed by the Kouretes Robocup team (http://www.kouretes.gr). Kouretes develop software that uses the Monas Robotic platform, which allows the integration of the robot capabilities as XML-specified Monas modules. Examples of these capabilities are vision, localisation, motion, and behaviour. These different capabilities/modules in the Monas architecture communicate with each other using the blackboard paradigm (Hayes-Roth, 1985). Thus, the $IAC2Monas$ tool included the definition of a new grammar for the transition expressions allowing for blackboard-based communication. The Kouretes Statechart Editor (KSE) tool (Topalidou-Kyniazopoulou et al., 2013) extended $IAC2Monas$ to provide a graphical CASE tool allowing for automatic code generation from the IAC model to the Nao robot API.

The *GGenerator* tool, also developed by the Kouretes Robocup team, allows to define generic agent behaviours using automatic framework-independent code generation, as long as the underlying framework is written in C++. This way a user can program physical (robots) or software agents that can be executed on any platform using any compatible software framework. The middle-ware for connecting to target platforms is a generic blackboard.

5 Evaluation

Evaluating an AOSE methodology is a difficult task according to a paper discussing the existing landscape (Sturm and Shehory, 2014). In that paper the authors list a number of techniques for evaluating a methodology or comparing it to alternatives. Among them are comparison/evaluation based on supported features, case studies and field experiments, lab experiments and surveys.

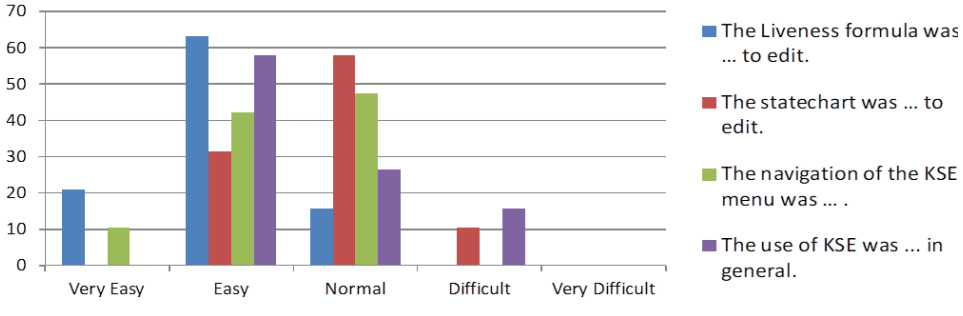
ASEME has been evaluated with lab experiments, where students used it for solving a well defined problem, field experiments and case studies, where the authors, or other independent researchers, have used ASEME to build real world systems. It can also be evaluated by the features that it supports.

5.1 Lab experiment and user satisfaction survey

To obtain an empirical evaluation of the KSE CASE tool for *ASEME* 28 students taking the autonomous agents class at an Electrical and Computer Engineering School of the Technical University of Crete were asked to use KSE and evaluate it in one of the two-hour laboratory sessions of the class. The students worked in small teams of two or three people per team. None of them had any prior experience with CASE tools, KSE, Monas, or RoboCup. This lab session was run three times to accommodate all students in the four available work stations.

The students first went through a quick tutorial on using KSE, which demonstrated the development of a Goalie behaviour for the Nao robot. This included the Gaia formulas for the goalie role, and its *IAC* model. Then, they were asked to use the existing functionalities of the Goalie (scan for the ball, kick the ball, approach the ball, etc.) to develop an *attacker* behaviour using KSE. Thus, the students did not have to develop the robot functionalities. They used KSE to define the attacker role's liveness and then to edit the statechart, i.e., to define variables and transition expressions. Looking at the goalie example they could find how to enable state transitions based on, e.g., the ball's position, or to use an action to set the value for, e.g., the direction for a kick. Then, they uploaded the software on the real robot and tested it. At the end of each lab session, a quick football game took place with the four developed attackers split in two teams of two players each.

Figure 16 Empirical evaluation of the KSE tool (CASE tool for ASEME) (see online version for colours)



Notes: The figures on the column correspond to the percentage of responders. For example, 20% of the responders found that 'the liveness formula was *very easy* to edit' (see the first column to the left). If there is no column this means that no responders gave this answer.

All student teams were able to deliver the requested attacker behaviour and enjoyed watching their players in the game. Then, the students were asked to fill in a questionnaire conceived to assess their satisfaction in using KSE, while obtaining information on their background as well. 19 students responded to the questionnaire. Only 21.05% stated that they were familiar with AOSE. Some of the most interesting results regarding their evaluation of KSE are presented in Figure 16. The most important findings of this study were the fact that most of the students found the KSE easy to use and that the concept of a liveness formula (unknown to the students before the lab) was

easy to understand and use; for more information the reader can consult the diploma thesis of Topalidou-Kyniazopoulou (2012). Nonetheless, a limitation of this study is that our approach was not directly compared to another.

5.2 Case studies and field experiments

$ASEME$ and its tools have been successfully used for the development of several real world systems, i.e., a situated product pricing agent [market-miner project (Spanoudakis and Moraitis, 2009a)], an ambient intelligence multi-agent system for knowledge-based and integrated services for mobility impaired users [ASK-IT project (Moraitis and Spanoudakis, 2007)], a wind turbine monitoring system (Smarsly and Hartmann, 2010), several applications in structural health monitoring (SHM) systems (Smarsly and Law, 2012), an ambient assisted living (AAL) system for the elderly and those suffering from mild cognitive impairment and Alzheimer disease [HERA project (Spanoudakis and Moraitis, 2015)], multi-agent system for the ubiquitous learning domain (Boudabous et al., 2018), and, for modelling the behaviour of robots (Topalidou-Kyniazopoulou et al., 2013). Actually, the Kouretes Robocup soccer team used $ASEME$ to model the behaviour of its robot players and won the second place in the SPL Open Challenge Competition in Robocup 2011 (Paraschos et al., 2012).

Finally, and with regard to the classical AI book of Russell and Norvig (2009) $ASEME$ has been applied in all types of environments, a result as yet unaccomplished by other methodologies. According to Russell and Norvig, environments:

a are partially to fully observable (depending on the omnipotence of the agent)

b may involve a single or multiple, cooperative or competitive, agents

c may be deterministic or stochastic (uncertain)

d may be episodic or sequential, based on whether the consequences of actions taken previously influence the future decisions of the agent

e can be static or dynamic (in the latter case the environment evolves while the agent deliberates)

f may be discrete or continuous, based on the way time is handled.

To address these environments, the use of the $ASEME$ *GGenerator* tool was demonstrated for the SimSpark 3D soccer simulation, and for the Wumpus World Simulator C++, a classic AI testbed (Papadimitriou et al., 2014). These two platforms are diverse and show the applicability of $ASEME$ for a partially observable, stochastic, dynamic, continuous, sequential, uncertain (noise), multi-agent (with both cooperative and competitive agents), with physical representation environment (SimSpark 3D, Kouretes soccer playing robot) and for a fully observable, deterministic, static, episodic, discrete, single agent environment with no uncertainty (Wumpus world). Of course, these two illustrative examples do not cover all possible combinations of environment types, but they model the extremes, and, thus, they give a clear indication of $ASEME$'s applicability.

6 Related work and conclusions

In this paper we presented a global view of $ASEME$, an AOSE methodology. Using this paper, a practitioner can discover a set of tools that will guide him to its use. $ASEME$ brings several innovations related to the state of the art (see e.g., Bresciani et al., 2004; Cossentino, 2005; DeLoach and Garcia-Ojeda, 2010; Garijo et al., 2005; Henderson-Sellers and Giorgini, 2005; Iglesias and Garijo, 2005; Padgham and Winikoff, 2004; Pavon et al., 2005; Picard and Gleizes, 2004; Wooldridge et al., 2000). It has been conceived as a model-driven development methodology and its models guide the developer from requirements analysis to code generation. $ASEME$ reuses and extends successful models of existing state of the art methodologies (e.g., Gaia, Tropos, MaSE, and UML). Engineers familiar with those will gain quick understanding of $ASEME$. The main advantages with other existing methodologies such as Bresciani et al. (2004), Cossentino (2005), DeLoach and Garcia-Ojeda (2010), Garijo et al. (2005), Henderson-Sellers and Giorgini (2005), Iglesias and Garijo (2005), Padgham and Winikoff (2004), Pavon et al. (2005), Picard and Gleizes (2004) and Wooldridge et al. (2000) are that $ASEME$:

- Is the first AOSE methodology to consider three levels of abstraction (i.e., society, agent and capability) in each development phase. Other methodologies (e.g., Cossentino, 2005; Padgham and Winikoff, 2004), usually start by defining the agents and their protocols in an analysis (or architectural design) phase and then focus in each agent in a design (or detailed design) phase. Moreover, it does not define a unique metamodel, as is usual with existing methodologies, but different metamodels for the different development phases capturing only the relevant elements each time.
- Defines agent architecture based on statecharts (Harel and Naamad, 1996), a well-known and general language, and does not make any other assumption concerning the architecture of the agents, giving this freedom to the designer. Other methodologies impose, or strongly imply, like Ingenias (Pavon et al., 2005), Prometheus (Jayatilleke et al., 2005; Padgham and Winikoff, 2004), Tropos (Bresciani et al., 2004), the mental attitudes-based agent architecture, e.g., BDI (Georgeff et al., 1999). Even though this can be useful for some applications, there are developers who want to use different agent architecture paradigms. Moreover, as there is no standardisation of agent architectures, to impose a specific model might discourage mainstream software developers from using agent technology.
- Automates the integration of EAC models in the IAC model by using the same language (i.e., of statecharts) for modelling both models in a uniform way. In the other AOSE methodologies this automation is not available or is still under study [see e.g., the current effort for integrating the AUML protocols in the Prometheus architecture (Abushark and Thangarajah, 2013)]. Using the orthogonality feature of statecharts, AMOLA allows to model agents that can concurrently participate into more than one protocol and integrate this ability with their other capabilities.
- Is applicable to both the software and physical (robot) agent development. Through shared variables it allows the different agent capabilities to exchange information.

- Can be used with agents with peer to peer (i.e., communicating with ACL messages exchange) or blackboard-based communication.
- Uniquely supports non-functional requirements, a feature only available in Tropos (Bresciani et al., 2004), however, not as a way to define alternative tasks that achieve the soft-goals as in Tropos, but as requirements that influence the way to implement a system (Pérez et al., 2006).
- Uniquely offers a process fragment for validating analysis phase models against non-functional requirements such as performance and scaling (Spanoudakis et al., 2018).

We are currently in the process of transferring the ASEME tools to the web. This will make it more accessible to practitioners that now have to install a specific Eclipse package and then some extensions and, finally, the ASEME dashboard. Moreover, the tooling will not need to be updated each time Eclipse releases a new version. Furthermore, we would like to add some features that would help the diagrams scale. For example, we aim to add the possibility to hide the use cases related to a role, or the use cases that one includes so that the SUC model is not cluttered after adding detail to several agent capabilities (Figure 6).

Another interesting path is towards solving the 'round-trip' problem, i.e., once a model is generated, the previous phase model is abandoned as it requires scarce and expensive resources to maintain. Round-trip engineering aims to convert the new model back into the previous phase model. This is particularly challenging as it requires an automated conversion that includes abstraction, a human capability (Selic, 2003).

In the future, the IAC model can be used by a new module of the agent which can keep track of the occurring transitions and detect anomalous or not frequent situations. For example, a broker agent [e.g., the one in Moraitis et al. (2005)] that keeps track of the web service invocation results suddenly realises that whenever it invokes a web service, it always gets a failure result, while normally it gets a failure in a small percentage of invocations. This could mean that its web service invocation component has failed, or it is outdated and needs an update. This meta-information on the agents executing lifecycle can be very useful if it can be automated in the agent's code generation. It can lead to self-healing and self-configuring, which are important capabilities in autonomic computing (Murch, 2004). Moreover, following the trend in software engineering of runtime models it would be interesting to see how the IAC could be a runtime model. Runtime models can be used for dynamic adaptation (Morin et al., 2009).

Numerous other directions exist for future work, the most challenging of all being to define programs that will edit the models themselves gradually leading to an implementation. It would be interesting to consider a knowledge-based approach for the SUC task decomposition process (i.e., to decompose a general task to specific ones). Another possibility could be to use evolutionary techniques to improve (or adapt) an IAC model. Goldsby et al. (2007) have proposed a method for evolving statecharts. It would be very interesting to see how the IAC model could evolve so as to keep its properties.

References

Abushark, Y. and Thangarajah, J. (2013) 'Propagating AUML protocols to detailed design', in Cossentino, M., Fallah Seghrouchni, A. and Winikoff, M. (Eds.): *1st Workshop on Engineering Multi-Agent Systems, Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg, Vol. 8245, pp.19–37.

Bellifemine, F.L., Caire, G. and Greenwood, D. (2007) *Developing Multi-Agent Systems with JADE*, Wiley Series in Agent Technology, Wiley-Blackwell, Chichester, UK.

Beydeda, S., Book, M. and Gruhn, V. (2005) *Model-Driven Software Development*, Springer, Berlin, Heidelberg.

Boudabous, S., Kazar, O. and Laouar, M.R. (2018) 'AMuL: the agents model for the U-learning system', in *Proceedings of the 8th International Conference on Information Systems and Technologies, ICIST'18*, ACM, pp.8:1–8:5.

Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F. and Mylopoulos, J. (2004) 'Tropos: an agent-oriented software development methodology', *Autonomous Agents and Multi-Agent Systems*, Vol. 8, No. 3, pp.203–236.

Cossentino, M. (2005) 'From requirements to code with the PASSI methodology', in Henderson-Sellers, B. and Giorgini, P. (Eds.): *Agent-Oriented Methodologies*, Chapter 4, pp.79–106, Idea Group Publishing, London, UK.

DeLoach, S.A. and Garcia-Ojeda, J.C. (2010) 'O-MaSE: a customisable approach to designing and building complex, adaptive multi-agent systems', *International Journal of Agent-Oriented Software Engineering*, Vol. 4, No. 3, pp.244–280.

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

García-Magariño, I., Rougemaille, S., Fuentes-Fernández, R., Migeon, F., Gleizes, M-P. and Gómez-Sanz, J. (2009) 'A tool for generating model transformations by-example in multi-agent systems', in Demazeau, Y., Pavón, J., Corchado, J.M. and Bajo, J. (Eds.): *7th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS 2009), Advances in Soft Computing*, Springer, Berlin, Heidelberg, Vol. 55, pp.70–79.

Garijo, F.J., Gomez-Sanz, J.J. and Massonet, P. (2005) 'The MESSAGE methodology for agent-oriented analysis and design', in Henderson-Sellers, B. and Giorgini, P. (Eds.): *Agent-Oriented Methodologies*, Chapter 8, pp.203–235, Idea Group Publishing, London, UK.

Gascueña, J.M., Navarro, E. and Fernández-Caballero, A. (2012) 'Model-driven engineering techniques for the development of multi-agent systems', *Engineering Applications of Artificial Intelligence*, Vol. 25, No. 1, pp.159–173.

Georgeff, M., Pell, B., Pollack, M., Tambe, M. and Wooldridge, M. (1999) 'The belief-desire-intention model of agency', in Müller, J.P., Rao, A.S. and Singh, M.P. (Eds.): *Intelligent Agents V: Agents Theories, Architectures, and Languages*, pp.1–10, Springer, Berlin, Heidelberg.

Goldsby, H.J., Knoester, D.B., Cheng, B.H., McKinley, P.K. and Ofria, C.A. (2007) 'Digitally evolving models for dynamically adaptive systems', in *International Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'07)*, IEEE, pp.13–22.

Gomez-Sanz, J.J., Fuentes-Fernández, R. and Pavón, J. (2011) 'Understanding agent oriented software engineering methodologies', in *12th International Workshop on Agent-Oriented Software Engineering*, Taipei.

Hahn, C., Madrigal-Mora, C. and Fischer, K. (2008) 'A platform-independent metamodel for multiagent systems', *Autonomous Agents and Multi-Agent Systems*, April, Vol. 18, No. 2, pp.239–266.

Harel, D. and Naamad, A. (1996) 'The STATEMATE semantics of statecharts', *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Vol. 5, No. 4, p.293.

Hayes-Roth, B. (1985) 'A blackboard architecture for control', *Artificial Intelligence*, Vol. 26, No. 3, pp.251–321.

Henderson-Sellers, B. and Giorgini, P. (2005) *Agent-Oriented Methodologies*, Idea Group Publishing, London, UK.

Iglesias, C.A. and Garijo, M. (2005) 'The agent-oriented methodology MAS-CommonKADS', in Henderson-Sellers, B. and Giorgini, P. (Eds.): *Agent-Oriented Methodologies*, Chapter 3, pp.46–78, Idea Group Publishing, London, UK.

Jayatilleke, G.B., Padgham, L. and Winikoff, M. (2005) 'A model driven component-based development framework for agents', *Comput. Syst. Sci. Eng.*, Vol. 20, No. 4, pp.273–282.

Kleppe, A.G., Warmer, J. and Bast, W. (2003) *MDA Explained: The Model Driven Architecture: Practice and Promise*, Addison-Wesley, Boston, MA, USA.

Moore, S.A. (2000) 'On conversation policies and the need for exceptions', in Dignum, F. and Greaves, M. (Eds.): *Issues in Agent Communication, Lecture Notes in Computer Science*, Springer, Heidelberg, Vol. 1916, pp.144–159.

Moraitis, P., Petraki, E. and Spanoudakis, N.I. (2005) 'An agent-based system for infomobility services', in Gleizes, M.P., Kaminka, G.A., Now, A., Ossowski, S., Tuyls, K. and Verbeeck, K. (Eds.): *Proceedings of the Third European Workshop on Multi-Agent Systems (EUMAS 2005)*, Koninklijke Vlaamse Academie van Belie voor Wetenschappen en Kunsten, pp.224–235.

Moraitis, P. and Spanoudakis, N. (2006) 'The GAIA2JADE process for multi-agent systems development', *Applied Artificial Intelligence*, Vol. 20, Nos. 2–4, pp.251–273.

Moraitis, P. and Spanoudakis, N. (2007) 'Argumentation-based agent interaction in an ambient-intelligence context', *IEEE Intelligent Systems*, Vol. 22, No. 6, pp.84–93.

Morin, B., Barais, O., Jezequel, J-M., Fleurey, F. and Solberg, A. (2009) 'Models@ Run.time to support dynamic adaptation', *Computer*, Vol. 42, No. 10, pp.44–51.

Murch, R. (2004) *Autonomic Computing*, IBM Press, Upper Saddle River, NJ.

Murray, J. (2004) 'Specifying agent behaviors with UML statecharts and StatEdit', in Polani, D., Browning, B., Bonarini, A. and Yoshida, K. (Eds.): *RoboCup 2003: Robot Soccer World Cup VII, Lecture Notes in Computer Science*, Springer, Heidelberg, Vol. 3020, pp.145–156.

Naur, P., Backus, J.W., Bauer, F.L., Green, J., Katz, C. and McCarthy, J. (1963) 'Revised report on the algorithmic language algol 60', *Communications of the ACM*, Vol. 6, No. 1, pp.1–17.

Nwana, H.S., Ndumu, D.T., Lee, L.C. and Collis, J.C. (1999) 'Zeus: a toolkit for building distributed multiagent systems', *Applied Artificial Intelligence*, Vol. 13, Nos. 1–2, pp.129–185.

OMG (2007) *Unified Modeling Language, Superstructure, V2.1.2*, Technical Report Formal/07-11-02, Object Management Group.

Padgham, L. and Winikoff, M. (2004) *Developing Intelligent Agent Systems: A Practical Guide*, Wiley Series in Agent Technology, Wiley, Chichester, UK.

Papadimitriou, G.L., Spanoudakis, N.I. and Lagoudakis, M.G. (2014) 'Extending the Kouretes Statechart Editor for generic agent behavior development', in Iliadis, L., Maglogiannis, I. and Papadopoulos, H. (Eds.): *Proceedings of the 10th International Conference on Artificial Intelligence Applications and Innovations (AIAI 2014), IFIP Advances in Information and Communication Technology*, Springer, Berlin, Heidelberg, Rhodes, Greece, 19–21 September, Vol. 436, pp.182–192.

Paraschos, A., Spanoudakis, N.I. and Lagoudakis, M.G. (2012) 'Model-driven behavior specification for robotic teams', in *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, Valencia, Spain, Vol. 1, pp.171–178.

Paurobally, S., Cunningham, J. and Jennings, N.R. (2004) 'Developing agent interaction protocols using graphical and logical methodologies', in Dastani, M.M., Dix, J. and El Fallah-Seghrouchni, A. (Eds.): *Programming Multi-Agent Systems, Lecture Notes in Computer Science*, Springer, Heidelberg, Vol. 3067, pp.149–168.

Pavon, J., Gomez-Sanz, J.J. and Fuentes, R. (2005) 'The INGENIAS methodology and tools', in Henderson-Sellers, B. and Giorgini, P. (Eds.): *Agent-Oriented Methodologies*, Chapter 9, pp.236–276, Idea Group Publishing, London, UK.

Pérez, J., Laguna, M.A., Crespo, Y. and González-Baixauli, B. (2006) 'Requirements variability support through MDD and graph transformations', in *International Workshop on Graph and Model Transformation (GraMoT05), Electronic Notes in Theoretical Computer Science*, Elsevier B.V., Tallinn, Vol. 152, pp.161–173.

Perini, A. and Susi, A. (2006) 'Automating model transformations in agent-oriented modelling', in Müller, J.P. and Zambonelli, F. (Eds.): *Agent-Oriented Software Engineering VI, Lecture Notes in Computer Science*, Springer, Heidelberg, Vol. 3950, pp.167–178.

Picard, G. and Gleizes, M-P. (2004) 'The ADELFE methodology', in Bergenti, F., Gleizes, M-P. and Zambonelli, F. (Eds.): *Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook*, Springer, Boston, MA, USA, pp.157–175.

Russell, S. and Norvig, P. (2009) *Artificial Intelligence: A Modern Approach*, 3rd ed., Prentice Hall Press, Upper Saddle River, NJ, USA.

Seidita, V., Cossentino, M. and Gaglio, S. (2009) 'Using and extending the SPEM specifications to represent agent oriented methodologies', in Luck, M. and Gomez-Sanz, J.J. (Eds.): *Agent-Oriented Software Engineering IX, Lecture Notes in Computer Science*, Springer, Vol. 5386, pp.46–59.

Selic, B. (2003) 'The pragmatics of model-driven development', *IEEE Software*, Vol. 20, No. 5, pp.19–25.

Smarsly, K. and Hartmann, D. (2010) 'Agent-oriented development of hybrid wind turbine monitoring systems', in Tizani, W. (Ed.): *Proceedings of ISCCBE International Conference on Computing in Civil and Building Engineering and the EG-ICE Workshop on Intelligent Computing in Engineering*, Nottingham University Press.

Smarsly, K. and Law, K.H. (2012) 'Advanced structural health monitoring based on multi-agent technology', in Zander, J. and Mostermann, P. (Eds.): *Computation for Humanity: Information Technology to Advance Society*, CRC Press – Taylor & Francis Group, LLC, Boca Raton.

Spanoudakis, N. (2009) *The Agent Systems Engineering Methodology (ASEME)*, PhD thesis, Paris Descartes University, Paris.

Spanoudakis, N. and Moraitis, P. (2008a) 'An agent modeling language implementing protocols through capabilities', in *2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, IEEE, Sydney, NSW, Australia, pp.578–582.

Spanoudakis, N. and Moraitis, P. (2008b) 'The agent modeling language (AMOLA)', in Dochev, D., Pistore, M. and Traverso, P. (Eds.): *Artificial Intelligence: Methodology, Systems, and Applications, Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg, Vol. 5253, pp.32–44.

Spanoudakis, N. and Moraitis, P. (2009a) 'Engineering an agent-based system for product pricing automation', *Engineering Intelligent Systems for Electrical Engineering and Communications*, Vol. 17, Nos. 2–3, pp.139–151.

Spanoudakis, N. and Moraitis, P. (2009b) 'Gaia agents implementation through models transformation', in Yang, J-J., Yokoo, M., Ito, T., Jin, Z. and Scerri, P. (Eds.): *Principles of Practice in Multi-Agent Systems, Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg, Vol. 5925, pp.127–142.

Spanoudakis, N. and Moraitis, P. (2010) 'Modular JADE agents design and implementation using ASEME', in *2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, IEEE, Toronto, Canada, Vol. 2, pp.221–228.

Spanoudakis, N. and Moraitis, P. (2011) 'Using ASEME methodology for model-driven agent systems development', in Weyns, D. and Gleizes, M-P. (Eds.): *Agent-Oriented Software Engineering XI, Lecture Notes in Computer Science*, Springer-Verlag, Berlin, Heidelberg, Vol. 6788, pp.106–127.

Spanoudakis, N. and Moraitis, P. (2015) 'Engineering ambient intelligence systems using agent technology', *IEEE Intelligent Systems*, Vol. 30, No. 3, pp.60–67.

Spanoudakis, N.I., Floros, E., Mitakidis, N. and Delias, P. (2018) 'Validating MAS analysis models with the ASEME methodology', *International Journal of Agent-Oriented Software Engineering*, Vol. 6, No. 2, pp.211–240.

Steinberg, D., Budinsky, F., Paternostro, M. and Merks, E. (2008) *Eclipse Modeling Framework*, 2nd ed., Addison-Wesley Professional, Boston, MA.

Sturm, A. and Shehory, O. (2014) 'The landscape of agent-oriented methodologies', in Shehory, O. and Sturm, A. (Eds.): *Agent-Oriented Software Engineering*, Chapter 7, pp.137–154, Springer, Berlin, Heidelberg.

Taentzer, G., Crema, A., Schmutzler, R. and Ermel, C. (2008) 'Generating domain-specific model editors with complex editing commands', in Schürr, A., Nagl, M. and Zündorf, A. (Eds.): *Applications of Graph Transformations with Industrial Relevance, Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg, Vol. 5088, pp.98–103.

Topalidou-Kyniazopoulou, A. (2012) *A CASE (Computer-Aided Software Engineering) Tool for Robot-Team Behavior – Control Development*, Diploma thesis, Technical University of Crete.

Topalidou-Kyniazopoulou, A., Spanoudakis, N.I. and Lagoudakis, M.G. (2013) 'A CASE tool for robot behavior development', in Chen, X., Stone, P., Sucar, L.E. and Zant, T. (Eds.): *RoboCup 2012: Robot Soccer World Cup XVI, Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg, Vol. 7500, pp.225–236.

Winikoff, M., Padgham, L., Harland, J. and Thangarajah, J. (2005) 'Declarative procedural goals in intelligent agent systems', in *Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR2002)*, Morgan Kaufmann Publishers, Toulouse, France, San Francisco, CA, 22–25 April.

Wooldridge, M., Jennings, N.R. and Kinny, D. (2000) 'The Gaia methodology for agent-oriented analysis and design', *Autonomous Agents and Multi-Agent Systems*, Vol. 3, No. 3, pp.285–312.

Wooldridge, M.J. (2009) *An Introduction to MultiAgent Systems*, John Wiley & Sons, Chichester, UK.