# 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
E-mail: nikos@amcl.tuc.gr

## Pavlos Moraitis

Laboratory of Informatics Paris Descartes (LIPADE)
University of Paris
45 rue des Saints-Peres
75006 Paris
France
E-mail: pavlos.moraitis@u-paris.fr

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 $ASEME$ (for $AgentSystemsEngineeringMEthodology$). Several parts of the methodology concerning different aspects of the whole development process have been already 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 by using $ASEME$ and which prove that $ASEME$ is very well suited for an easy development or real world applications.

**Keywords:** Software engineering; Agent Oriented Software Engineering; Intelligent agents; Multi-Agent Systems; Methodologies; Software process

## 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 that 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 (may perform such tasks that, when performed by humans, we consider as evidence of a certain intelligence) (see e.g. [45]). 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 [17] for implementing Gaia models [44] using the JADE agent platform [2]. It emerged as an evolution of the Gaia2JADE process influenced by the requirements analysis phase of Tropos [5] and the work of Moore [15] on conversation policies. Existing papers in the literature have focused on specific aspects of the methodology [31, 32, 34, 35, 36]. 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 [3], 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.

In the area of Agent-Oriented Software Engineering (AOSE) a number of development methodologies has been proposed during the last 15 years (e.g. [5, 6, 7, 8, 12, 13, 21, 24, 26, 44]). Although some methodologies, e.g. Tropos [25], have proposed MDE processes for some phases of the development process, $ASEME$ 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 modeling languages and having most of the MAS part of the required code being automatically generated. Moreover, the design phase model of $ASEME$ is a statechart [10], a successful modeling paradigm well known to engineers, that can be implemented using a variety of programming languages or, as we will show herein, an agent-oriented framework. The models that are used by $ASEME$ are defined by the Agent Modeling Language ($AMOLA$). Both the inter- and intra-agent control are defined using the statechart formalism that allows for seamless integration of agent capabilities and interaction protocols.

## 2 ASEME Method Concepts

Until today, a number of AOSE methodologies have been proposed, each supporting different styles of agent programming and different agent architectures (for a survey see [40]).

$ASEME$ is a methodology for developing autonomous agents and multi-agent systems. It uses the Agent Modeling Language ($AMOLA$) for modeling 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. It supports a modular agent design approach and introduces the concepts of intra- and inter-agent control. 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 modeling of the intra and inter-agent control is based on statecharts (see e.g. [10]). 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 *intra-agent control* 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 [21]). $AMOLA$ describes both an agent and a multi-agent system. $AMOLA$ is compatible with the Object Management Group's (OMG) Model Driven Architecture (MDA) paradigm [14]. 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 SAG model
- The platform independent model (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 intra-agent control model (IAC)
- The Platform Specific Model 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 Fig. 1 using the extended `SPEM 2.0` language for representing agent oriented methodologies [28].

The $ASEME$ process is iterative, allowing for incremental development. In ASEME, the SAG, SRM, IAC and a Platform Specific Model (PSM) 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 Fig. 1 (i.e. those that point clockwise form 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.

Three levels of abstraction are defined for each phase. The first is the *societal level*. There, the whole multi-agent system behavior is modeled. 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

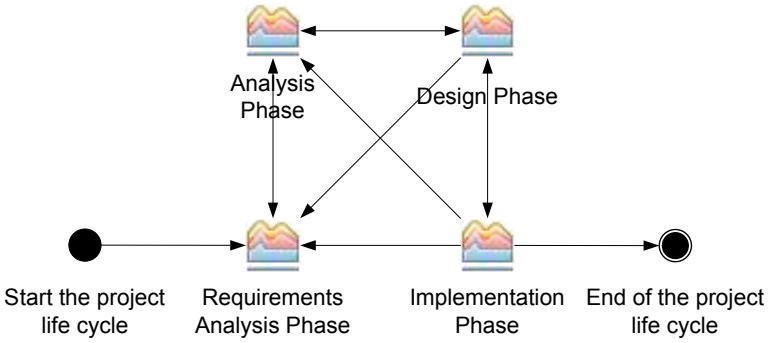


**Figure 1** ASEME Process Overview.

| *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, Functionality Graph (FG) |
| **Design** | *Society Control* intEr-Agent Control (EAC), Ontology, Message Types | *Agent Control* Intra-Agent Control (IAC) | *Components* IAC |
| **Implementation** | Platform management code | Agent code | Capabilities code |

**Figure 2** ASEME phases and their AMOLA products.

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 Fig. 2, the $ASEME$ phases, the different levels of abstraction and the models related to each one of them are presented. 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 [12]. This system's

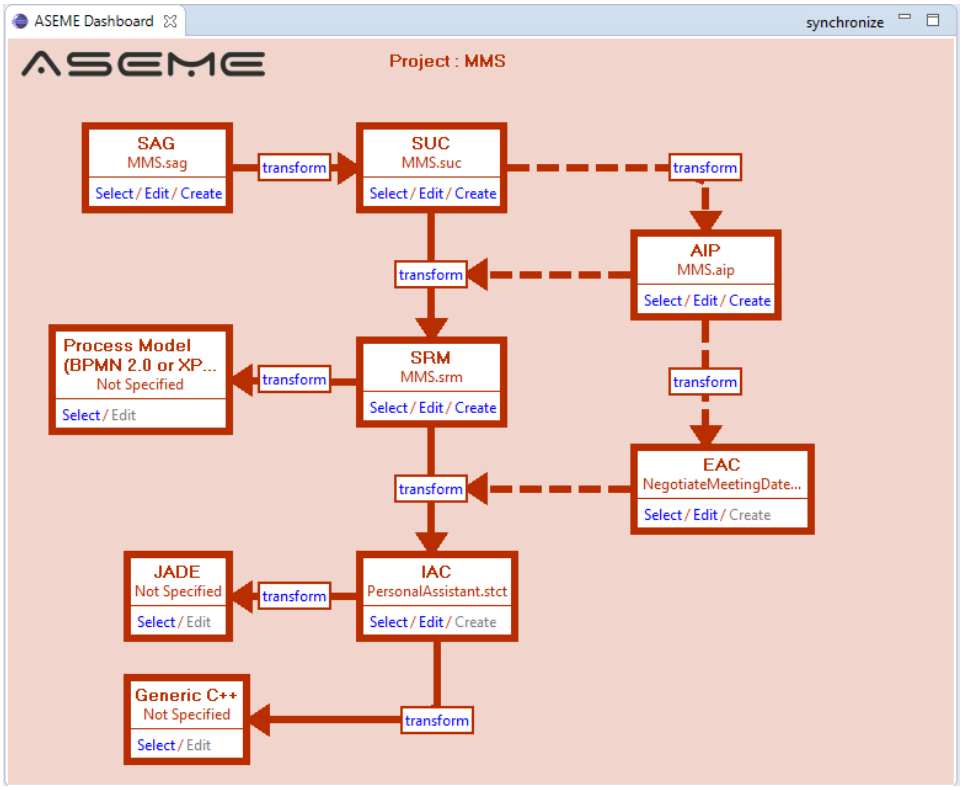


**Figure 3** The $ASEME$ dashboard.

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 organization 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 [41]. It is depicted in Fig. 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 modeling, which must also be used when developing multi-agent systems. 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.

### *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

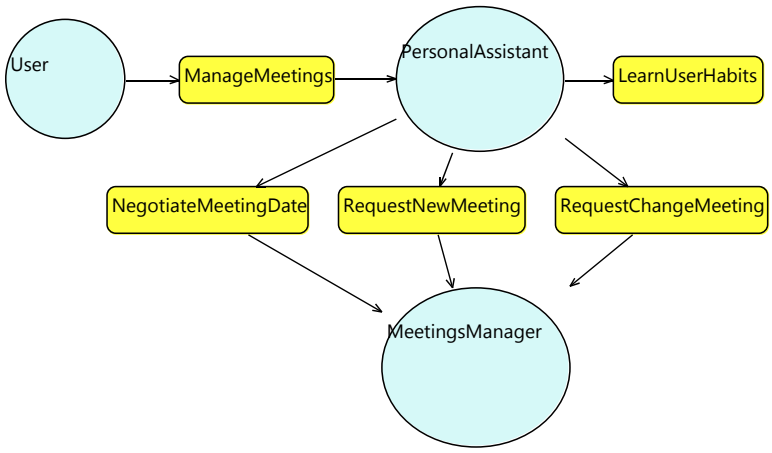


**Figure 4** The System Actors-Goals model.

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 organization that will develop the software) together with a firm representative (who represents the client).

### *3.1.1 System Actors and Goals Model ($SAG$)*

The $AMOLA$ model for the requirements analysis phase is the $SAG$ model, a graph involving actors and goals. A goal of one actor (owner of the goal) may be dependent for its realization 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. Summarizing, 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 Fig. 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 10 seconds".

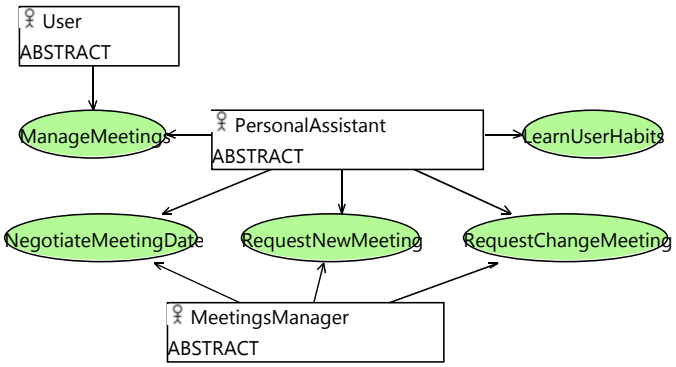


**Figure 5** The initial System Use Cases model (transformed from Fig. 4).

### *3.2 Analysis Phase*

The first task of the analysis phase is to transform the $SAG$ model to a System Use Case model ($SUC$ model).

#### *3.2.1 The System Use Cases Model ($SUC$)*

The System Use Cases Model ($SUC$) is similar to the use case diagram of UML [1]. It helps to visualize the system in terms of roles and tasks that they realize. 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 initialized automatically by the $SAG$ model. The *SAG2SUC* transformation maps concepts from the $SAG$ model to those of the $SUC$ model. Fig. 5 shows the produced $SU$C 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".

The analyst 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 analyst can choose to create more roles and define interactions between them. In the agent level, the analyst will eventually decompose the general use cases (as the "learn user habits" of our example) to more elementary ones.

Fig. 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 a "Human" role.

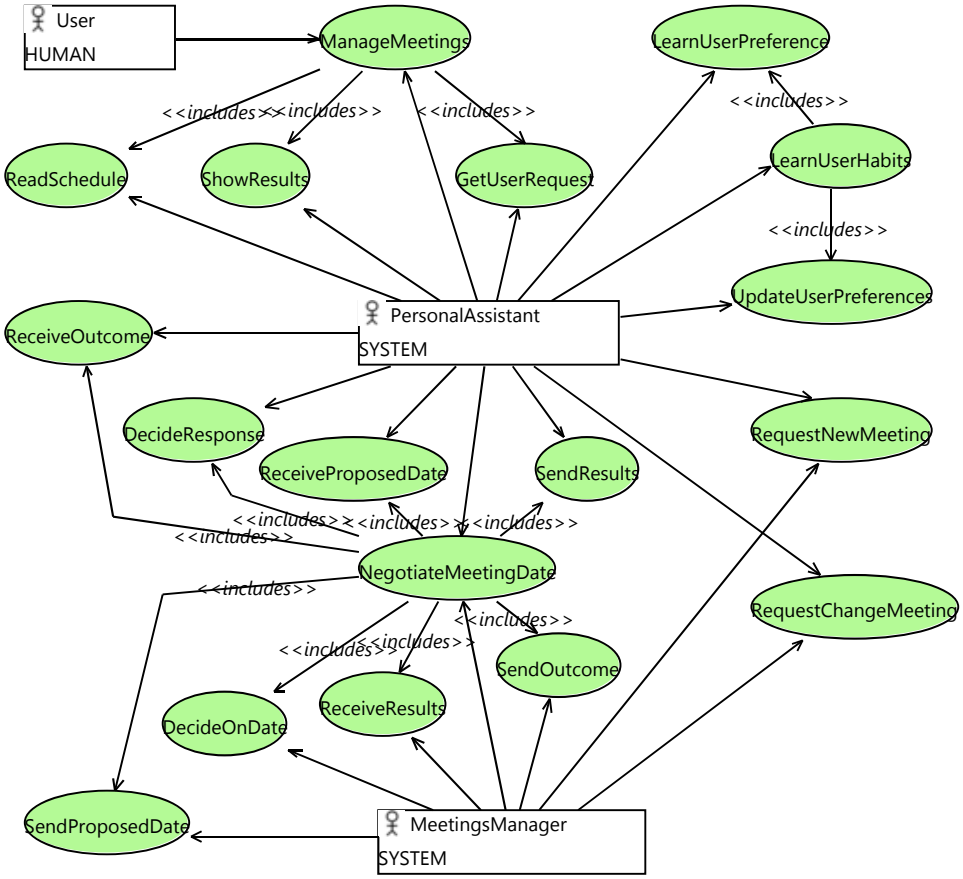


**Figure 6** The refined SUC diagram.

### *3.2.2 The Agent Interaction Protocols Model (AIP)*

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

- Protocol
  - name (a String type property for storing the name of the Protocol)
  - participants (a list of two or more references to Participant instances)
- Participant
  - name (a String type property for storing the name of the Participant)
  - engaging_rules (a String type property for storing the specifications of the prerequisites for the participant to enter the protocol in free text format)
  - outcomes (a String type property for storing the possible outcomes of the protocol in free text format)
  - 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 behavior of the role. It connects all the role's activities using the Gaia operators [44]. Briefly, $A.B$ means that activity $B$ is executed after activity $A$, $A^{\tilde{}}$ 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 ($\omega$) operator of Gaia with the tilde ($\tilde{}$) as it is more practical and quickly available in most keyboards.

The $AIP$ model is automatically initialized from the $SUC$ model. One protocol is created for every use case that has more than one role participants. The $SUC2AIP$ transformation tool initializes the process part of each protocol participant adding all the included use cases connected with the "OP?" symbol. This is a general feature that characterizes 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 Fig. 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. Fig. 8 shows the properties again after they have been edited by the modeler.

### *3.2.3 The Systems Roles Model ($SRM$)*

The system roles model ($SRM$) is mainly inspired by the Gaia roles model [44]. A role model is defined for each $SUC$ system role and contains the following elements

- Activity
  - name (a String type property for storing the name of the Activity)
  - functionality (the functionality related to this activity)

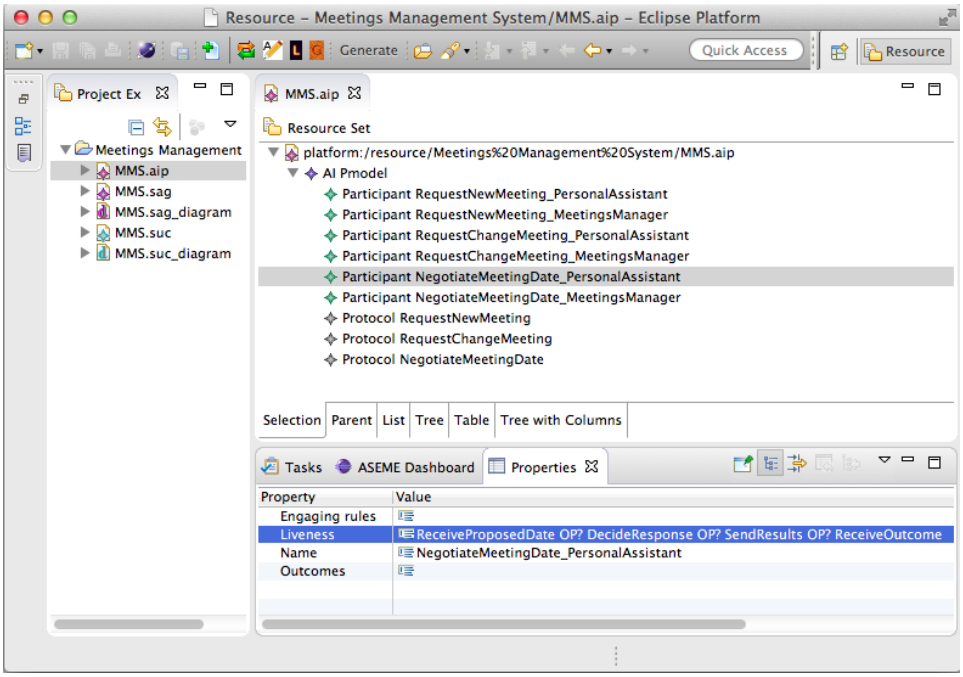


**Figure 7** The automatically generated AIP model

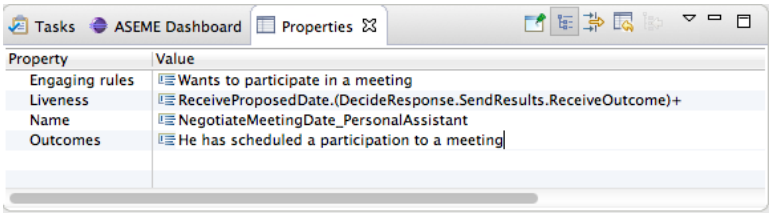


**Figure 8** The refined *PersonalAssistant* participant of the *NegotiateMeetingDate* protocol

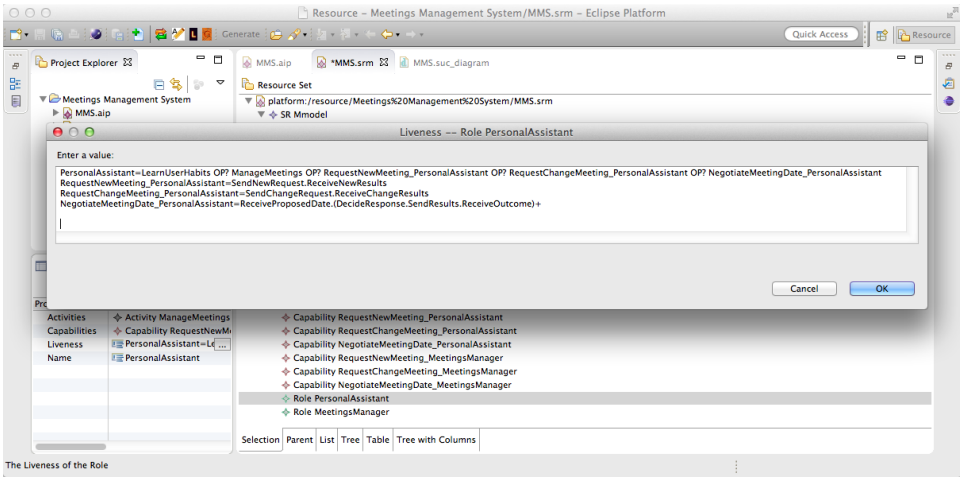


**Figure 9** The generated *PersonalAssistant* role in the SRM model. The edit dialog shows the liveness property of the role.

- Capability
  - name (a String type property for storing the name of the Capability)
  - activities (a list of zero or more references to Activity instances)
- Role
  - name (a String type property for storing the name of the Role)
  - capabilities (a list of zero or more references to Capability instances)
  - activities (a list of zero or more references to Activity instances)
  - liveness (the process that the role follows in the form of liveness formulas)

Therefore, a role aggregates capabilities and capabilities aggregate activities. 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 system roles model ($SRM$). 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 Fig. 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 Fig. 8). Notice that on the right hand side of the first formula participate all the use cases connected with the personal assistant role except those that are included by them. Behind the liveness formula 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).

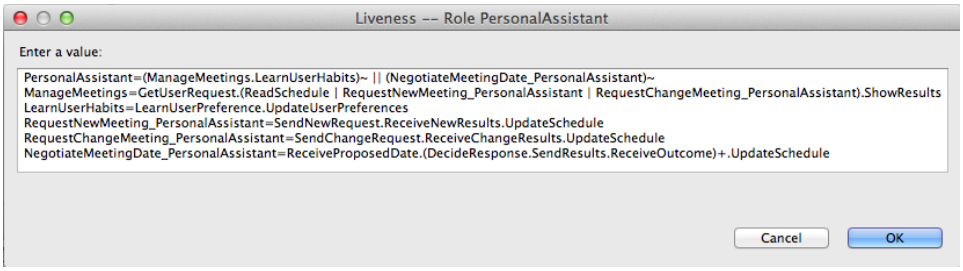


**Figure 10** The refined liveness formula of the *PersonalAssistant* role in the SRM.

After the $SUC2SRM$ transformation, the analyst refines the liveness formulas, connecting capabilities and activities with the appropriate Gaia operators (see the refined $SRM$ model in Fig. 10). During the refinement process, the analyst can choose to add activities to the protocol parts in the liveness formula but he has to keep the imported part from the $AIP$ intact. There is a reason for this restriction when building the liveness formulas. An agent must comply with a protocol specification. This means that it must do his part of the protocol as it is specified. The agent is free to determine how to achieve the tasks specified by the protocol, e.g. expand one protocol activity in a following formula.

The functionality graph ($FG$) is a graphical view of the $SRM$, where the analyst associates each activity participating in the liveness formulas of the $SRM$ to the technology or tool (functionality) that it will use (see Fig. 11 for the capabilities of the personal assistant). This is the point where the analyst proposes the use of a platform for instantiation, e.g., in our example, the JADE framework that adheres to the FIPA standards. This strategic choice also defines the programming language that will be used, in this case Java. This association of activities to functionalities is documented in the $SRM$ model and the functionality property of the activity.

Depending on the development iteration the functionalities can be vague and abstract (like the "machine learning" functionality) or concrete and final (like the "JADE FIPA AMS" and "argumentation-based decision making" functionalities).

In the presented $FG$, for example, it has been decided that the persistent information will be stored in files. Another analyst might opt for a more professional approach, such as the use of a database, or more sophisticated one, such as the use of the JADE Persistence package. Such choices are evident in the FG and the development team can have a meeting over this and discuss the alternatives. At the end of the day, the FG indicates the competences needed by the software development team. In our knowledge ASEME is the only AOSE methodology that provides this facility to assist technical project and risk management.

### *3.3 Design Phase*

The ASEME design phase process is presented in Fig. 12. The three activities reflect the three different levels of abstraction in the software development. In the society level we have the definition of the inter-agent control 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 input, while the reverse indicates an output).

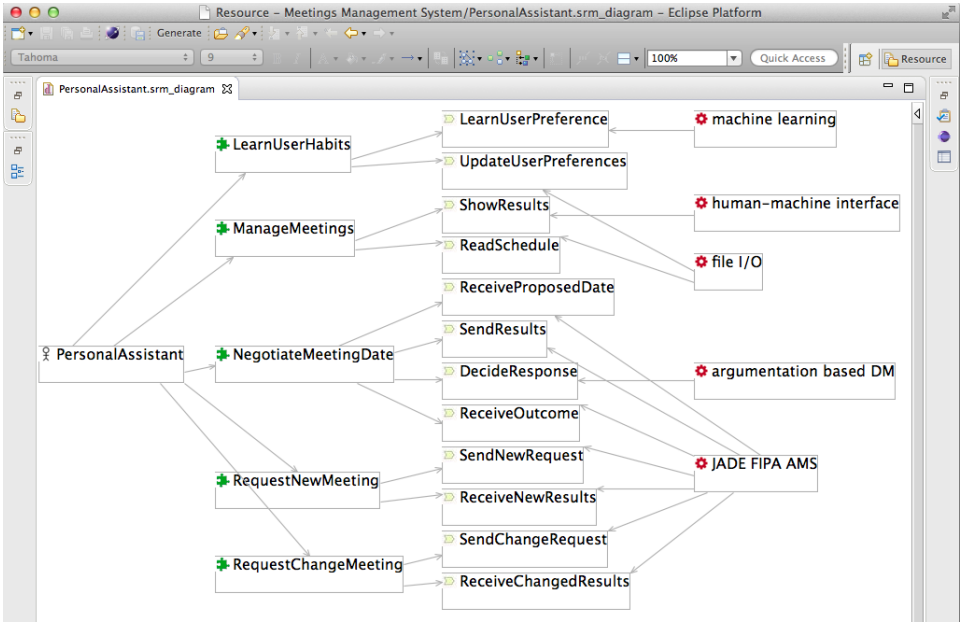


**Figure 11** The Functionality Graph for the *PersonalAssistant* role of the MMS.

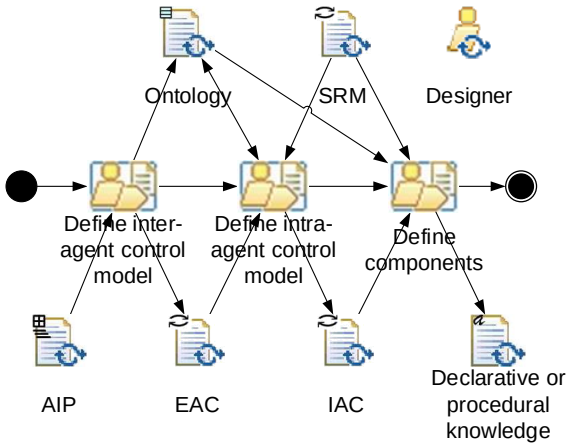


**Figure 12** The ASEME Design Phase.

The agents communicate using interaction protocols that are described by the *inter-agent control (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 *intra-agent control (IAC)* concept. The first activity ("define inter-agent control model") consists of four tasks and produces three models, namely the EAC, the performatives and the ontology.

### *3.3.1 The intEr-Agent Control Model (EAC)*

The inter-agent control is defined as a statechart [10]. It is initialized by transforming the agent interaction protocols of the analysis phase to statecharts. Statecharts are used for modeling 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 behavioral 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 behavioral aspects of a system is its greatest advantage [18]. 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 inter-agent control model in the society level of abstraction, and the intra-agent control model (IAC) 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 labeled 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.

The latter case is also true for a transition with no expression. Note that each state automatically starts its activity on entrance. In ASEME, we have used a logic-based language for representing transition expressions [7], however, a designer can use the representation language he/she prefers, taking into account the statechart language syntax and semantics. A message event is expressed by $P(x, y, c)$ where $P$ is the performative, $x$ is the sender role, $y$ the receiver role and $c$ the message body.

The items that the designer can use for defining the state transition expressions are the message performatives, the ontology used for defining the messages content and the timers. Timers can be defined as normal variables initializing them to a value representing

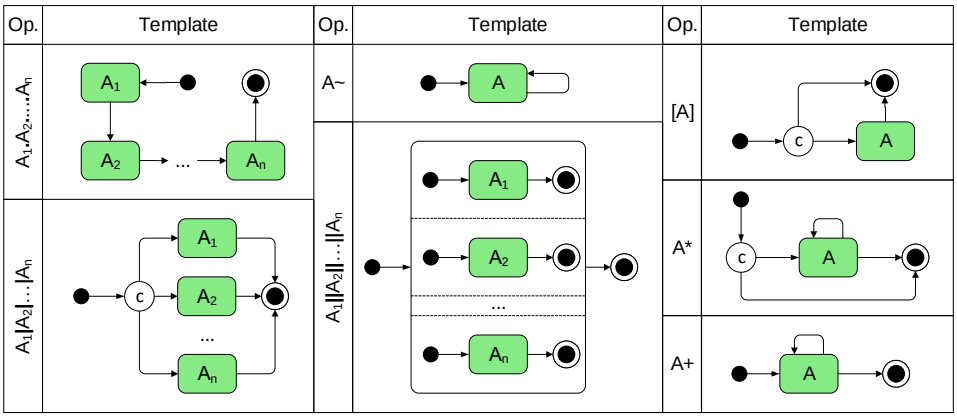


**Figure 13** Templates of extended Gaia operators for Statechart generation.

the number of milliseconds until they timeout (at which time their value is equal to zero). The transition expressions can use the timeout unary predicate, which is evaluated to true if the timer value is equal to zero, and false otherwise. Timers are initialized in the action part of a transition expression, while the timeout predicate can be used in both the event and condition parts of the transition expression depending on the needs of the designer.

Having defined the statechart as it is used in AMOLA it is now possible to proceed to the definition of the inter-agent control (EAC) model. The EAC is a statechart that contains an initial (START) state, an AND-state named after the protocol and a final (END) state. The AND-state contains as many OR-states as the protocol roles named after the roles. Two transitions connect the START state to the AND state and the AND state to the END state.

Here the reader should note that the inter-agent control model does not impose a specific way for interpreting the exchanged messages or a technology for exchanging them. These issues are defined by the developers according to the platform that they will use for deploying their system and to their expertise. For example, in FLBC [5] the effects of a request message are linked to the beliefs of the sender which may not be the case in another communication language with different semantics. Thus, a procedural agent might not have a model of beliefs in contrast with a BDI, i.e. belief-desire-intention, one [**?** ].

The AIP2EAC tool transforms the process part of the agent interaction protocol model to the inter-agent control model (EAC), which is a statechart. 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 Fig. 13.

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:

$$NegotiateMeetingDate = MeetingsManager||PersonalAssistant$$

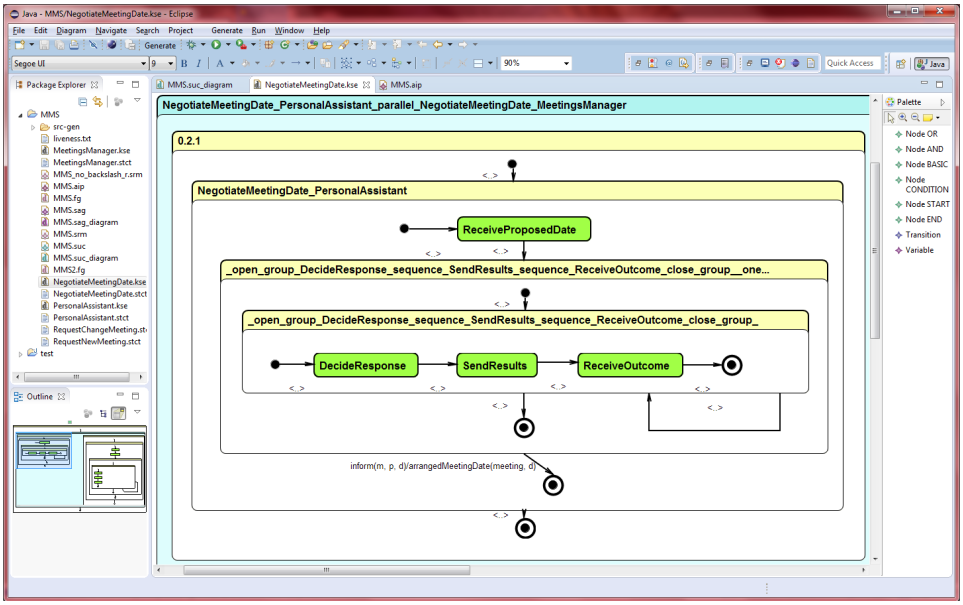


**Figure 14** The EAC model as it was produced by the AIP2EAC transformation with one transition expression added. Screenshot taken from a computer with Windows 7, Eclipse Modeling Tools Luna R2 and ASEME v.2.1.

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

After applying the transformation algorithm, the statechart depicted in Fig. 14 is created. The *ReceiveProposedDate* state is a basic state (drawn as a green-colored rounded rectangle), while the *NegotiateMeetingDate_PersonalAssistant* state is an OR state (drawn as a yellow-color-labeled 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.

Then, the designer defines the message performatives 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 parameters (also referred to as the ontology), the timers and the message contents (also part of the ontology).

Finally, in the last task of the "Define Inter-agent Control Model" ASEME activity, the transition expressions are defined (see [31] for the details). The preconditions of the agent interaction protocol become the conditions of a transition from a START state that targets the first state of the protocol for each role. In Fig. 14 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 transistion expression does not have a condition.

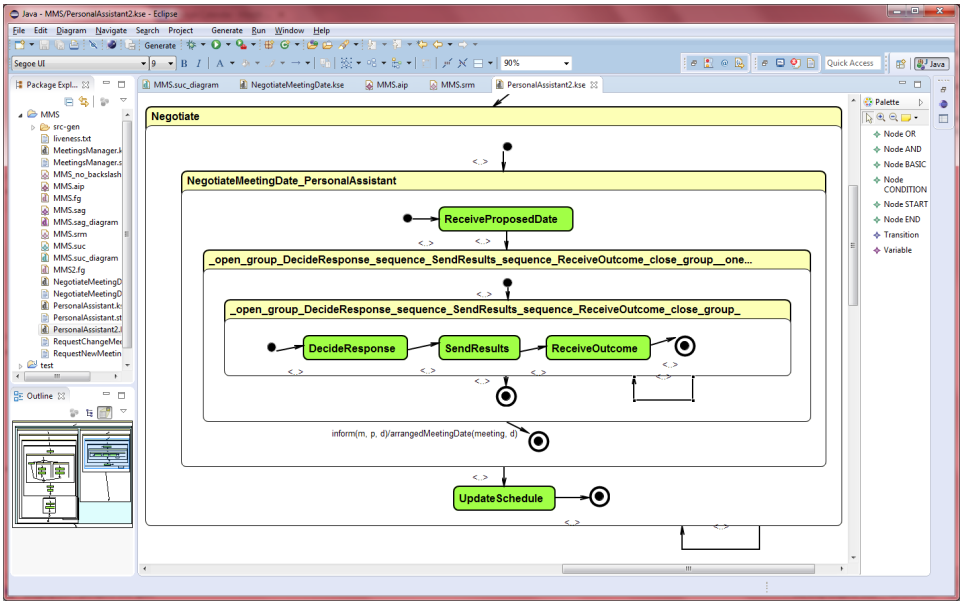


**Figure 15** The IAC model produced with the SRM2IAC transformation.

In the agent level, the intra-agent control (IAC) is created using statecharts in the same way with the inter-agent control model (EAC). The difference is that the top level state (root) corresponds to the modeled agent (which is named after the agent type). One IAC is defined for each agent type. The intra-agent control is initialized 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 Fig. 13.

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 Fig. 13. If one of the states is further refined in a next formula, then new substates are defined for it in a recursive way. Fig. 15 presents the IAC model that is produced by the transformation process (SRM2IAC) if its input is the refined SRM shown in Fig. 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 Fig. 10).

The transition expressions from the inter-agent control (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 Fig. 15 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 "Intra-agent control model" that lists all the activities as states. The output depends on the technology used for each activity and can be declarative or procedural knowledge (or both).

### *3.4 Implementation Phase*

The implementation phase's goal is to transform the platform independent model to a platform specific model. 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 CASE (Computer-Aided Software Engineering) 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 [35]. 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 behavior 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 Fig. 16 the reader can see on the left hand side the Java classes automatically generated when the user hit 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 [35]. In Fig. 16 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) [39] and they are freely available from `github.com` (on-line software project hosting using the git revision control system). The interested reader can be guided to downloading binaries and sources from the ASEME project web-site (`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 [34], liveness2BPMN [38], IAC2JADE [35], IAC2Monas [23], and GGenerator [22], and a CASE tool, KSE [43].

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

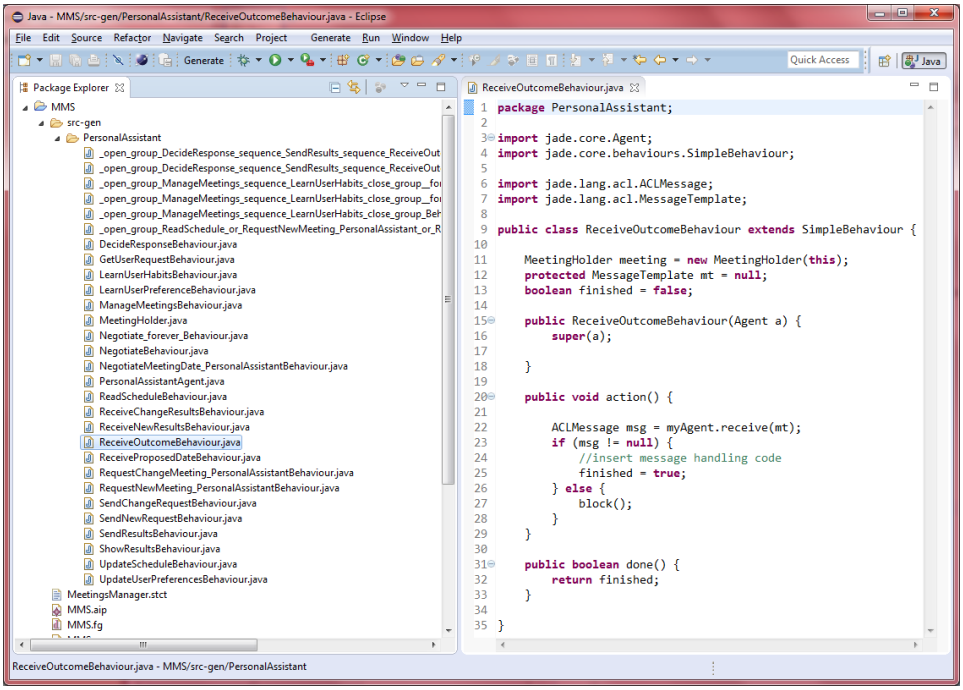


**Figure 16** The *PersonalAssistant* java package automatically generated by the IAC model by the IAC2JADE transformation.

to any platform through a generic blackboard interface [22], and the Business Process Modeling 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 [18], the authors used the liveness formulas of SRM to automatically create the process model of a service protocol [38]. They showed how a modeler can use the process model to detect flows in the system analysis and design and verify the system's behavior 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 10 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 3 seconds without problem, by adding more agent instances of a specific type [38].

The IAC2Monas tool was developed by the Kouretes Robocup team (`http://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, localization, motion, and behavior. These different capabilities/modules in the Monas architecture communicate with each other using the blackboard paradigm [11]. 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 [43] 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 defining generic agent behaviors 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 [40]. 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 we asked 28 students taking the Autonomous Agents class at a Technical University (name disclosed due to the double blind review process) to use KSE and evaluate it in one of the 2-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 behavior for the Nao robot. 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 behavior using KSE. Thus, the students did not have to develop the robot functionalities. They used KSE to define the roles liveness and then edit the statechart (define variables and transition expressions). 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.

All student teams were able to deliver the requested Attacker behavior 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 Fig. 17. Most of the students found the KSE use to be easy and we found out that the concept of a liveness formula (unknown to the students before the lab) was easy to understand and use.

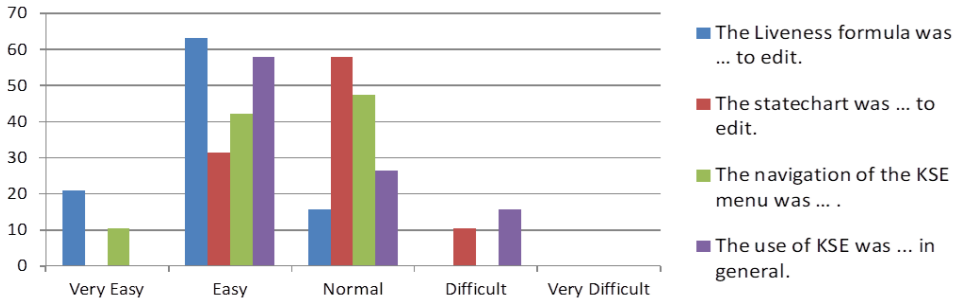


**Figure 17** Empirical evaluation of the KSE tool (CASE tool for ASEME). The figures on the column correspond to the percentage of responders.

### *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 [33]), an ambient intelligence multi-agent system for knowledge-based and integrated services for mobility impaired users (ASK-IT project [18]), a wind turbine monitoring system [29], several applications in Structural Health Monitoring systems (SHM) [30], an Ambient Assisted Living (AAL) system for the elderly and those suffering from mild cognitive impairment and Alzheimer disease (HERA project [37]), a multi-agent system for the ubiquitous learning domain [4], and, for modeling the behavior of robots [43]. Actually, the Kouretes Robocup soccer team used ASEME to model the behavior of its robot players and won the second place in the SPL Open Challenge Competition in Robocup 2011 [23].

Finally, and with regard to the classical AI book of Russell and Norvig [27] ASEME has been applied in all types of environments, a result as yet unaccomplished other methodologies. In the work of Papadimitriou et al. [22], the transparent use of the GGenerator was demonstrated for the SimSpark 3D soccer simulation and the classic AI testbed, Wumpus World Simulator C++. 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).

## 6 Conclusion

In this paper we presented 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. 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, UML). Engineers familiar with those will gain quick understanding of ASEME.

All in all, we have come a long way; however, there exist multiple challenges that call for further work. Having defined protocols and allowed agents to incorporate them in their capabilities the next challenge is to regulate the ability of an agent to participate in a protocol. Several authors in the literature have worked on organization rules and norms for regulating agent-based interactions, e.g. [46]. The question is whether this is a design issue, a runtime issue, or both.

The intra-agent control model (IAC) can be used by a new module of the agent which can keep track of the occurring transitions and detect anomalous or not fre-quent situations. For example, a broker agent (e.g. the one in [16]) that keeps track of the web service invocation results suddenly realizes that whenever it invokes a web service he always gets a failure result, while normally he 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 [20]. Moreover, following the trend in software engineering of runtime models we would like to see how the IAC could be a runtime model. Runtime models can be used for dynamic adaptation [19].

Numerous other directions exist, the most challenging of all being to define programs that will edit the models themselves gradually leading to an implementation. We would consider a knowledge-based approach for the SUC (e.g. to decompose a general task to specific ones) or SRM (e.g. to assign a functionality to an activity). Another possibility could be to use evolutionary techniques to improve (or adapt) an IAC model. Goldsby et al. [9] 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

[1] Unified Modeling Language, Superstructure, V2.1.2. Technical Report formal/07-11-02, Object Management Group, 2007.

[2] F. L. Bellifemine, G. Caire, and D. Greenwood. *Developing Multi-agent Systems with JADE*. Wiley Series in Agent Technology. Wiley-Blackwell, 2007.

[3] S. Beydeda, M. Book, and V. Gruhn, editors. *Model-Driven Software Development*. Springer, Berlin Heidelberg, 2005.

[4] S. Boudabous, O. Kazar, and M. R. Laouar. AMuL: The Agents Model for the U-learning System. In *Proceedings of the 8th International Conference on Information Systems and Technologies*, ICIST '18, pages 8:1–8:5. ACM, 2018.

[5] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos. Tropos: An Agent-Oriented Software Development Methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.

[6] M. Cossentino. From Requirements to Code with the PASSI Methodology. In B. Henderson-Sellers and P. Giorgini, editors, *Agent-Oriented Methodologies*, chapter IV, pages 79–106. Idea Group Publishing, 2005.

[7] S. A. DeLoach and J. C. Garcia-Ojeda. O-MaSE: a customisable approach to designing and building complex, adaptive multi-agent systems. *International Journal of Agent-Oriented Software Engineering*, 4(3):244–280, 2010.

[8] F. J. Garijo, J. J. Gomez-Sanz, and P. Massonet. The MESSAGE Methodology for Agent-Oriented Analysis and Design. In B. Henderson-Sellers and P. Giorgini, editors, *Agent-Oriented Methodologies*, chapter VIII, pages 203–235. Idea Group Publishing, 2005.

[9] H. J. Goldsby, D. B. Knoester, B. H. Cheng, P. K. McKinley, and C. A. Ofria. Digitally Evolving Models for Dynamically Adaptive Systems. In *International Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '07)*, pages 13–22. IEEE, 2007.

[10] D. Harel and A. Naamad. The STATEMATE Semantics of Statecharts. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 5(4):293, 1996.

[11] B. Hayes-Roth. A blackboard architecture for control. *Artificial Intelligence*, 26(3):251–321, 1985.

[12] B. Henderson-Sellers and P. Giorgini, editors. *Agent-Oriented Methodologies*. Idea Group Publishing, 2005.

[13] C. A. Iglesias and M. Garijo. The Agent-Oriented Methodology MAS-CommonKADS. In B. Henderson-Sellers and P. Giorgini, editors, *Agent-Oriented Methodologies*, chapter III, pages 46–78. Idea Group Publishing, 2005.

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

[15] S. A. Moore. On Conversation Policies and the Need for Exceptions. In F. Dignum and M. Greaves, editors, *Issues in Agent Communication*, volume 1916 of *Lecture Notes in Computer Science*, pages 144–159. Springer, Heidelberg, 2000.

[16] P. Moraitis, E. Petraki, and N. I. Spanoudakis. An Agent-based System for Infomobility Services. In M. P. Gleizes, G. A. Kaminka, A. Now, S. Ossowski, K. Tuyls, and K. Verbeeck, editors, *Proceedings of the Third European Workshop on Multi-Agent Systems (EUMAS 2005)*, pages 224–235. Koninklijke Vlaamse Academie van Belie voor Wetenschappen en Kunsten, 2005.

[17] P. Moraitis and N. Spanoudakis. The GAIA2JADE Process for Multi-Agent Systems Development. *Applied Artificial Intelligence*, 20(2-4):251–273, 2006.

[18] P. Moraitis and N. Spanoudakis. Argumentation-Based Agent Interaction in an Ambient-Intelligence Context. *IEEE Intelligent Systems*, 22(6):84–93, 2007.

[19] B. Morin, O. Barais, J.-M. Jezequel, F. Fleurey, and A. Solberg. Models@ Run.time to Support Dynamic Adaptation. *Computer*, 42(10):44–51, 2009.

[20] R. Murch. *Autonomic Computing*. IBM Press, 2004.

[21] L. Padgham and M. Winikoff. *Developing Intelligent Agent Systems: A Practical Guide*. Wiley Series in Agent Technology. Wiley, 2004.

[22] G. L. Papadimitriou, N. I. Spanoudakis, and M. G. Lagoudakis. Extending the Kouretes Statechart Editor for Generic Agent Behavior Development. In L. Iliadis, I. Maglogiannis, and H. Papadopoulos, editors, *Proceedings of the 10th International Conference on Artificial Intelligence Applications and Innovations (AIAI 2014), Rhodes, Greece, September 19-21*, volume 436 of *IFIP Advances in Information and Communication Technology*, pages 182–192. Springer Berlin Heidelberg, 2014.

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

[24] J. Pavon, J. J. Gomez-Sanz, and R. Fuentes. The INGENIAS Methodology and Tools. In B. Henderson-Sellers and P. Giorgini, editors, *Agent-Oriented Methodologies*, chapter IX, pages 236–276. Idea Group Publishing, 2005.

[25] A. Perini and A. Susi. Automating Model Transformations in Agent-Oriented Modelling. In J. P. Müller and F. Zambonelli, editors, *Agent-Oriented Software Engineering VI*, volume 3950 of *Lecture Notes in Computer Science*, pages 167–178. Springer, Heidelberg, 2006.

[26] G. Picard and M.-P. Gleizes. The ADELFE Methodology. In F. Bergenti, M.-P. Gleizes, and F. Zambonelli, editors, *Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook*, pages 157–175. Springer US, Boston, MA, 2004.

[27] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009.

[28] V. Seidita, M. Cossentino, and S. Gaglio. Using and Extending the SPEM Specifications to Represent Agent Oriented Methodologies. In M. Luck and J. J. Gomez-Sanz, editors, *Agent-Oriented Software Engineering IX*, volume 5386 of *Lecture Notes in Computer Science*, pages 46–59. Springer, 2009.

[29] K. Smarsly and D. Hartmann. Agent-Oriented Development of Hybrid Wind Turbine Monitoring Systems. In W. Tizani, editor, *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, 2010.

[30] K. Smarsly and K. H. Law. Advanced Structural Health Monitoring based on Multi-Agent Technology. In J. Zander and P. Mostermann, editors, *Computation for Humanity: Information Technology to Advance Society*. CRC Press – Taylor & Francis Group, LLC, Boca Raton, 2012.

[31] N. Spanoudakis and P. Moraitis. An Agent Modeling Language Implementing Protocols through Capabilities. In *2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, pages 578–582, Sydney, NSW, Australia, 2008. IEEE.

[32] N. Spanoudakis and P. Moraitis. The Agent Modeling Language (AMOLA). In D. Dochev, M. Pistore, and P. Traverso, editors, *Artificial Intelligence: Methodology,*

*Systems, and Applications*, volume 5253 of *Lecture Notes in Computer Science*, pages 32–44. Springer Berlin Heidelberg, 2008.

[33] N. Spanoudakis and P. Moraitis. Engineering an agent-based system for product pricing automation. *Engineering Intelligent Systems for Electrical Engineering and Communications*, 17(2-3):139–151, 2009.

[34] N. Spanoudakis and P. Moraitis. Gaia Agents Implementation through Models Transformation. In J.-J. Yang, M. Yokoo, T. Ito, Z. Jin, and P. Scerri, editors, *Principles of Practice in Multi-Agent Systems*, volume 5925 of *Lecture Notes in Computer Science*, pages 127–142, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[35] N. Spanoudakis and P. Moraitis. Modular JADE Agents Design and Implementation Using ASEME. In *2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, volume 2, pages 221–228, Toronto, Canada, 2010. IEEE.

[36] N. Spanoudakis and P. Moraitis. Using ASEME Methodology for Model-Driven Agent Systems Development. In D. Weyns and M.-P. Gleizes, editors, *Agent-Oriented Software Engineering XI*, volume 6788 of *Lecture Notes in Computer Science*, pages 106–127. Springer-Verlag, Berlin, Heidelberg, 2011.

[37] N. Spanoudakis and P. Moraitis. Engineering ambient intelligence systems using agent technology. *IEEE Intelligent Systems*, 30(3):60–67, 2015.

[38] N. I. Spanoudakis, E. Floros, N. Mitakidis, and P. Delias. Validating MAS analysis models with the ASEME methodology. *International Journal of Agent-Oriented Software Engineering*, 6(2):211–240, 2018.

[39] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks. *Eclipse Modeling Framework (2nd Edition)*. Addison-Wesley Professional, 2 edition, 2008.

[40] A. Sturm and O. Shehory. The Landscape of Agent-Oriented Methodologies. In O. Shehory and A. Sturm, editors, *Agent-Oriented Software Engineering*, chapter 7, pages 137–154. Springer Berlin Heidelberg, 2014.

[41] G. Taentzer, A. Crema, R. Schmutzler, and C. Ermel. Generating Domain-Specific Model Editors with Complex Editing Commands. In A. Schürr, M. Nagl, and A. Zündorf, editors, *Applications of Graph Transformations with Industrial Relevance*, volume 5088 of *Lecture Notes in Computer Science*, pages 98–103. Springer Berlin Heidelberg, 2008.

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

[43] A. Topalidou-Kyniazopoulou, N. I. Spanoudakis, and M. G. Lagoudakis. A CASE Tool for Robot Behavior Development. In X. Chen, P. Stone, L. E. Sucar, and T. Zant, editors, *RoboCup 2012: Robot Soccer World Cup XVI*, volume 7500 of *Lecture Notes in Computer Science*, pages 225–236. Springer Berlin Heidelberg, 2013.

[44] M. Wooldridge, N. R. Jennings, and D. Kinny. The Gaia Methodology for Agent-Oriented Analysis and Design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.

[45] M. J. Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley & Sons, 2009.

[46] F. Zambonelli, N. R. Jennings, and M. Wooldridge. Developing multiagent systems: The Gaia methodology. *ACM Transactions on Software Engineering and Methodology*, 12(3):317–370, 2003.