PASSIM: a simulation-based process for the development of multi-agent systems

Massimo Cossentino

ICAR/CNR
Viale delle Scienze
90128 Palermo, Italy
E-mail: cossentino@pa.icar.cnr.it

Giancarlo Fortino*, Alfredo Garro, Samuele Mascillaro and Wilma Russo

Dipartimento di Elettronica Informatica e Sistemistica (DEIS)
Università della Calabria
Via P. Bucci, 87036 Rende (CS), Italy
E-mail: g.fortino@unical.it
E-mail: garro@unical.it
E-mail: samuele.mascillaro@deis.unical.it
E-mail: w.russo@unical.it
*Corresponding author

Abstract: This paper presents the Process for Agent Specification, Simulation and Implementation (PASSIM), a simulation-based development process for Multi-agent Systems (MASs), which was obtained by integrating the well-known and established Process for Agent Societies Specification and Implementation (PASSI) methodology and a Statecharts-based simulation methodology supporting functional and nonfunctional validation of the MAS being developed. PASSIM can be effectively used as an experimental tool in the context of Agent-Oriented Software Engineering (AOSE) for evaluating the benefits of using simulation for MAS development. To exemplify this process and demonstrate its effectiveness, a case study concerning the analysis, design and simulation of a complex MAS implementing an agent-based e-marketplace is defined and detailed.

Keywords: Multi-agent Systems; MASs; Agent-Oriented Software Engineering; AOSE; discrete-event simulation; method engineering.

Reference to this paper should be made as follows: Cossentino, M., Fortino, G., Garro, A., Mascillaro, S. and Russo, W. (2008) 'PASSIM: a simulation-based process for the development of multi-agent systems', *Int. J. Agent-Oriented Software Engineering*, Vol. 2, No. 2, pp.132–170.

Biographical notes: Massimo Cossentino obtained his Master's degree in Electronics Engineering and his PhD in Computer Science Engineering from the University of Palermo. He has been a Researcher of the Italian National Research Council from 2001. In 2007 he was an invited Associate Professor at the University of Belfort-Montbelliard (UTBM). He is currently researching on Agent-Oriented Software Engineering (AOSE), more specifically on agent-oriented design methodologies (he is the main author of

PASSI, a methodology for designing MASs), multi-agent systems metamodels, agent patterns and design tools. He is the author of about 80 papers published in international journals and proceedings of conferences/workshops. He has chaired and co-chaired several scientific events and coordinated the activity of several research/standardisation groups such as the FIPA Methodology Technical Committee (2003) and the Agentlink III AOSE Technical Forum Group (2004–2007).

Giancarlo Fortino is an Associate Professor of Computer Science at the Department of Electronics, Computer and Systems Science (DEIS) of the University of Calabria, Rende (CS), Italy. He received his Laurea degree in Computer Science Engineering from the University of Calabria in 1995 and a PhD in Computer Science and Systems Engineering from the same institution in 2001. In 1997 and 1999 he was Visiting Researcher at the International Computer Science Institute (ICSI), Berkeley (CA), USA. His current research interests are agent-based systems, agent oriented software engineering, streaming content distribution networks, wireless sensor networks, distributed multimedia systems and GRID systems. He is the author of more than 80 papers in international journals, books and conferences. He is a member of IEEE, IEEE Computer and Communications Society, and ACM.

Alfredo Garro received his Laurea degree in Computer Science Engineering and his PhD in Computer Science and Systems Engineering from the University of Calabria (Italy). From 1999 to 2001, he worked at CSELT (today TILAB), the Telecom Italia Group R&D Laboratories. He is currently an Assistant Professor of Computer Science at the Department of Electronics, Computer and System Sciences of the University of Calabria. His research interests include agent-oriented software engineering, game theory, logic programming, and knowledge representation and reasoning. His list of publications contains about 40 papers published in international journals, books and proceedings of international and national conferences.

Samuele Mascillaro received his Laurea degree in Computer Science Engineering from the University of Palermo (Italy) in 2006. He is a fellowship PhD student in Computer Science and Systems Engineering at the Department of Electronics, Computer and System Sciences, University of Calabria (Italy). His research interests are in the field of agent-oriented software engineering and, in particular, are focused on the definition of agent-based modelling techniques and tools for the development of complex software systems.

Wilma Russo received her Laurea degree in Physics from the University of Naples (Italy). She then moved to the University of Calabria (Italy), where she held the position of Associate Professor of Computer Science starting 1986. After a period with the University of Salerno, she returned to the University of Calabria, where she is now full Professor of Computer Science at the Department of Electronics, Informatics and Systems (DEIS). Her current research interests are mainly focused on parallel and distributed computing and systems, agent oriented software engineering, agent based modeling and simulation, internet computing and content distribution networks. She is the author of more than 80 papers in international journals, books and conferences.

1 Introduction

Simulation is widely applied in many industrial fields, such as aerospace, automotive or energy production, but its application in the support of software products and processes is still underestimated to date. Despite its limited exploitation in software engineering, simulation has been recognised to be an effective tool for supporting software engineering experimentations involving requirements management, project management, training, process improvement, architecture and Component Off-The-Shelf (COTS) integration, product-line practices, risk management and acquisition management (Christie, 1999; Mayrhauser, 1993).

With the emergence of Agent-Oriented Software Engineering (AOSE) as a new discipline (Luck *et al.*, 2004; Bernon *et al.*, 2005) which aims at identifying and defining models and techniques suitable for the development of complex software systems in terms of Multi-agent Systems (MASs), we wonder if simulation could play a more strategic role in the development of MASs than that which it played in the development of traditional and/or conventional software systems, and, more specifically, if simulation could provide a substantial added value when applied to support the development process of MASs (Uhrmacher, 2002).

The answer to our first question lies in the complexity of MASs with respect to the traditional software systems. A MAS is a system composed of several agents, capable of reaching goals that are difficult to be achieved by an individual system (Wooldridge, 2002). MASs can manifest self-organisation and complex behaviours even when the individual strategies of all their agents are simple. Thus, the use of simulation can be crucial in the analysis of the MAS being developed at different scales of observation (macro, micro and meso levels) (Zambonelli and Omicini, 2004) and, also, for the discovery of emergent properties which were not taken into account or were not considered at all in the design phase.

To answer the second question we need to quantify the claimed added value in using simulation for MAS development through actual experimentations covering the whole software development life cycle of MASs: requirements capture, analysis, design, implementation, deployment and testing. To date a few MAS development processes have been proposed in the literature, such as Electronic Institutions (Sierra *et al.*, 2004), DynDEVS/James (Rohl and Uhrmacher, 2004), CaseLP (Martelli *et al.*, 1999), GAIA/MASSIMO (Fortino *et al.*, 2005b), TuCSon/pi (Gardelli *et al.*, 2005) and Joint Measure (Sarjoughian *et al.*, 2001), which incorporate simulation to support the design phase of the MAS development life cycle, with the main focus on the validation and performance evaluation of the designed MAS model. However, to quantify the benefits of using simulation for MAS development, further research work needs to be carried out in the aforementioned direction and in further directions encompassing all the phases of the MAS development life cycle. The major benefits would be product quality improvement and project risk minimisation. These would derive from the use of simulation in pinning down MAS requirements early in the development life cycle, in testing out alternate modifications of requirements, in safely examining alternate architectures and designs, and in gaining insights into timing, resource usage and bottlenecking.

In this paper we propose the Process for Agent Specification, Simulation and Implementation (PASSIM), a simulation-based process for the development of MASs which incorporates a simulation phase for the prototyping of the MAS being developed and for functional and nonfunctional validation. PASSIM was obtained by integrating method fragments coming from two existing agent-oriented methodologies according to a *process-driven* method engineering approach (Fortino *et al.*, 2005b; Cossentino *et al.*, 2007). A method fragment is a portion of a software development process which has two fundamental elements (Brinkkemper *et al.*, 1999):

1 (work) products and their structures

2 procedures and their execution order for developing (work) products.

In particular, PASSIM was obtained by integrating method fragments from the Process for Agent Societies Specification and Implementation (PASSI) methodology (Cossentino, 2005) for carrying out the analysis, design and coding phases, and the Distilled State Charts (DSC)-based simulation method (Fortino *et al.*, 2005a–b) for supporting the simulation phase.

PASSIM is exemplified through a case study concerning the analysis, design and simulation of a MAS which represents an Agent-based e-Marketplace (AeM). In particular, the simulation phase allows for the functional validation of AeM being developed and for the performance evaluation of different types of agents in terms of completion time for searching and buying a product. The rest of the paper is organised as follows: In Section 2 PASSIM is presented by describing how it was obtained through an experiment of situational method engineering and by overviewing each of its composing phases. Section 3 shows the adaptation process of some activities and related work products of the design phase based on PASSI and the simulation phase based on the DSCs. Section 4 proposes a complete case study concerning the modelling and simulation of an AeM. Section 5 discusses some related agent-based design and development approaches incorporating simulation. Finally, conclusions are drawn and directions for future research briefly elucidated.

2 A process for agent specification, simulation and implementation

PASSIM is an agent-oriented software development process that uses simulation for prototyping the MAS being developed and validating requirements. The creation of the PASSIM process was carried out through the composition of parts coming from two existing methodologies: PASSI (Cossentino, 2005) and the DSC-based simulation methodology (Fortino *et al.*, 2005b). The composition of this new process can be regarded as an experiment of Situational Method Engineering (SME) (Harmsen and Brinkkemper, 1995), which is currently supported by several approaches in the literature (Brinkkemper *et al.*, 1996; Henderson-Sellers, 2003; Ralyté and Rolland, 2001; Fortino *et al.*, 2005b; Cossentino *et al.*, 2007). In particular, PASSIM was created according to a process-driven approach (Fortino *et al.*, 2005b; Cossentino *et al.*, 2007) which involves:

- *The choice or the definition of a software development life cycle suitable for the specific problem and for the specific application domain.*

An iterative-incremental life cycle was chosen which is partly also derived from Royce's final waterfall model (Royce, 1970) and specifically introduces the simulation phase to validate the system design before coding. In particular, the chosen life cycle is articulated into five phases (see Figure 1):

1 Requirements Specification

2 Design

3 Simulation

4 Coding

5 Deployment.

After the *Simulation* phase, the designers can either proceed with the remaining part of the process, if they want to implement the software's final release, or use the results of the simulation to feedback on the *Design* phase and/or the *Requirements Specification* phase.

- *The selection of suitable method fragments for carrying out each phase of the chosen software development life cycle. Method fragments can be derived from already existing methodologies or from ad hoc defined ones.*

  Table 1 reports the method fragments which were selected from both the PASSI methodology and the DSC-based simulation methodology for carrying out each phase of the chosen software development life cycle of Figure 1. For each method fragment the table shows the related activities and delivered work products. The selection of these fragments was easily performed since all the method fragments of the two exploited methodologies were available and ready to use. The obtained software development process (PASSIM) consists of five phases carried out by six different method fragments.

- *The adaptation of method fragments in order to allow their integration in the new methodology.*

  The *DSC-based Simulation* method fragment has been modified to take as input the work products produced by the *Agent Implementation* method fragment, selected from PASSI. In particular, the modified version of the method fragment translates the structural and dynamic diagrams produced by the *Agent Implementation* fragment into a MAS model based on DSC.

Figure 1 The software development life cycle of PASSIM

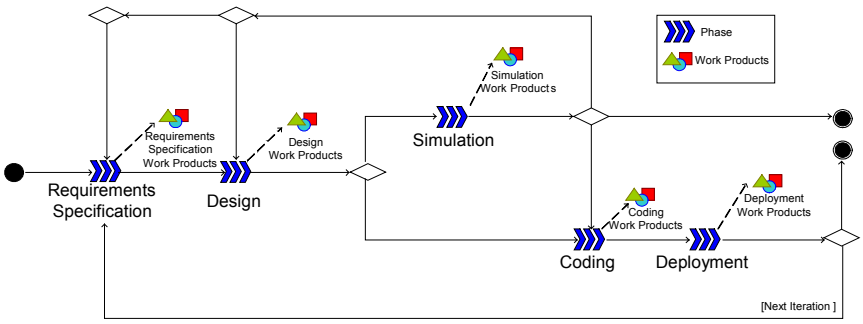


Table 1 The method fragments of PASSIM

Phase	*Composed method fragment*	*Atomic method fragments*	*Work product (kind)*	*Source methodology*
Requirements Specification	System requirements	Domain description	Domain description diagram (use-case diagram)	PASSI
		Agents identification	Agents identification diagram (use-case diagram)	
		Roles identification	Roles identification diagrams (sequence diagram)	
		Tasks specification	Tasks specification diagrams (activity diagram)	
Design	Agent society	Domain ontology description	Domain ontology description diagram (class diagram)	PASSI
		Communication ontology description	Communication ontology description diagram (class diagram)	
		Roles description	Roles description diagram (class diagram)	
		Protocols description	Protocols description (sequence diagram)	
	Agent implementation	Agent structure definition	Single-agent structure definition diagrams (class diagram)	PASSI
		Agent behaviour description	Multi-agent structure definition diagram (class diagram)	
			Single-agent behaviour description diagrams (activity/state diagram)	
			Multi-agent behaviour description diagram (activity/state diagram)	
Simulation	DSC-based simulation	Simulation model definition	Multi-agent System Distilled StateChart simulation model (MASDSC diagram)	DSC-based simulation
		MAS code generation	MAS Code (C(MASDSC) diagram)	
		Simulation implementation	Simulator program	
		Simulation execution	Simulation results	
Coding	Code	Code reuse	Code for the target agent platform	PASSI
		Code refinement		
Deployment	Deployment	Deployment configuration	Deployment diagrams	PASSI

The phases of PASSIM, which are carried out by the method fragments selected from PASSI, are fully supported by the PASSI Toolkit (PTK), developed as a Rational Rose plug-in, whereas the *Simulation* is supported by a DSC Visual Toolset (Fortino *et al.*, 2007), developed as an Eclipse plug-in, which allows for DSC-based visual modelling and automatic code generation.

Although our concept of method fragments is related to the OMG SPEM Process Component (SPEM, 2006), we adopt the method fragment meta-model defined in Cossentino *et al.* (2007) where each method fragment is composed of:

- a portion of a process that delivers a significant work product (*e.g.*, a class, a sequence diagram, a system analysis document including several diagrams and the description text)
- the produced work product
- some preconditions (such as required inputs)
- a list of components of the MAS meta-model that are defined/refined by the work done in the fragment
- some guidelines (best practices on how to perform the prescribed work)
- a glossary of terms used in the fragment
- some composition guidelines (for reusing the fragment in a new process)
- aspects of the fragment regarding specific application fields
- dependency relationships with other fragments.

Since method fragments can be composed of finer-grained ones, we introduce here three different levels of granularity for them:

1 smaller fragments (also called *Atomic*) which deliver diagram-size work products

2 bigger fragments (also called *Composed*) which deliver complex documents obtained by the composition of several smaller work products in order to obtain a larger scope document

3 phases collecting several composed fragments belonging to the same conceptual/design area.

The list of fragments used in PASSIM is reported in Table 1. Composed method fragments are clustered in the System Requirements, Design, Simulation, Coding and Deployment phases. In the following subsections each phase of PASSIM is described with reference to the method fragments selected for carrying it out.

2.1 Requirements Specification

The Requirements Specification phase is carried out by the *System Requirements* method fragment selected from PASSI, which produces a model of system requirements in terms of agency and purpose. This method fragment is composed of four atomic fragments: *Domain (Requirements) Description*, *Agents Identification*, *Roles Identification* and *Tasks Specification*.

The *Domain Description* produces a use-case diagram that represents actors and use cases (a functional description of the system) identified for the system using a hierarchical decomposition if it is required by the problem complexity. In the *Agents Identification*, agents are identified by assigning a responsibility to each agent for a part of the functionalities of the whole system; this fragment produces a use-case diagram, called Agents Identification diagram (AId). In particular, the designer clusters some of the use cases within a package and gives it the name of the agent that will be responsible for accomplishing the specific functionalities of the clustered use cases. Once all the use cases have been assigned to the identified agents, the designer can define scenarios in which the agents will be involved (*Roles Identification*). Such scenarios are modelled through a set of UML sequence diagrams which show that each agent may be involved in several different activities and may appear more than once in each scenario playing different roles. Finally, in the *Tasks Specification*, the tasks of each agent are specified through UML activity diagrams.

2.2 Design

The Design phase is carried out by two (composed) method fragments extracted from PASSI: the *Agent Society* and the *Agent Implementation*.

2.2.1 The Agent Society fragment

The *Agent Society* composed method fragment includes four atomic method fragments: *Domain Ontology Description*, *Communication Ontology Description*, *Roles Description*, and *Protocols Description*.

In the *Domain Ontology Description* the design of the domain ontology is performed by means of a class diagram (DOD diagram) that describes the ontology in terms of concepts (categories, entities of the domain), predicates (assertions on properties of concepts) and actions (performed in the domain). This diagram can also be regarded as an XML schema that can be used to obtain a Resource Description Framework (RDF) (FIPA, 2001; RDF, 1999) which encodes the ontological structure.

The *Communication Ontology Description* produces a class diagram (COD diagram) that shows all the agents and all their communications (relationships among agents). This diagram is drawn on the basis of the AId (see Section 2.1). A class is introduced for each agent, and an association is introduced for each communication between two agents. As communication is a way to exchange knowledge, it is also important to introduce the proper data structure (coming from the entities described in the DOD diagram) in each agent. The association line that represents each communication is drawn from the initiator of the conversation to the other agent (participant) as can be deduced from the description of their interaction performed in the *Roles Identification*. Each communication is characterised by three attributes, (Ontology, Agent Interaction Protocol and Content Language), which are grouped into an association class. The roles, initially identified in the *Agents Identification*, are completely defined in the *Roles Description* that produces a UML class diagram in which classes are used to represent roles. In particular, each role uses several elementary tasks to implement its complex behaviour and, finally, roles are grouped in packages representing agents.

The *Protocols Description* is required only when the FIPA standard protocols are not sufficient to solve some communication problems and new protocols must be introduced.

2.2.2 The Agent Implementation fragment

The *Agent Implementation* method fragment is composed of two different atomic fragments, each of them carried out at both the multi- and single-agent level of abstraction. The multi-agent level models the overall structure of the system (MAS structure and behaviour, interagent communications, *etc.*). The single-agent level of abstraction focuses on the implementation details of each agent.

In particular, the following two atomic method fragments are carried out at the multi- and single-agent levels:

- *Agent Structure Definition* (ASD), which uses conventional class diagrams to describe the structure of agents (represented by classes) and produces both the *Single-Agent Structure Definition* (SASD) diagrams and the *Multi-Agent Structure Definition* (MASD) diagram
- *Agent Behaviour Description* (ABD), which uses activity diagrams or statecharts to describe the behaviour of agents and produces the *Single-Agent Behaviour Description* (SABD) diagrams and the *Multi-Agent Behaviour Description* (MABD) diagram.

The MASD diagram represents the multi-agent system from the structural point of view. Agents are represented as classes with their behaviours in the operation compartment and attributes specifying the agent knowledge.

The agent behaviour at the multi-agent level is described by the MABD diagram. This is a UML activity diagram used to illustrate the dynamics of the system during the agents' life cycle. In this diagram, the involved agents and their tasks are represented with swim-lanes, operations are displayed as activities, and transitions among activities represent events like method invocations (when relating activities in the same swim-lane), new behaviour instantiations/invocations (when relating activities of the same agent but in different swim-lanes) or messages (when activities from two different agents are involved).

In the SASD diagram one class diagram is used for depicting the internal structure of each agent. This is a very detailed diagram, reporting attributes and methods of both the agent class and the classes of the tasks. The details of the behaviour of each agent are specified in the SABD diagram.

2.3 Simulation

The simulation phase is carried out by a (composed) method fragment, *DSC-based Simulation*, which is composed of four atomic method fragments: *Simulation Model Definition*, *MAS Code Generation*, *Simulation Implementation* and *Simulation Execution*.

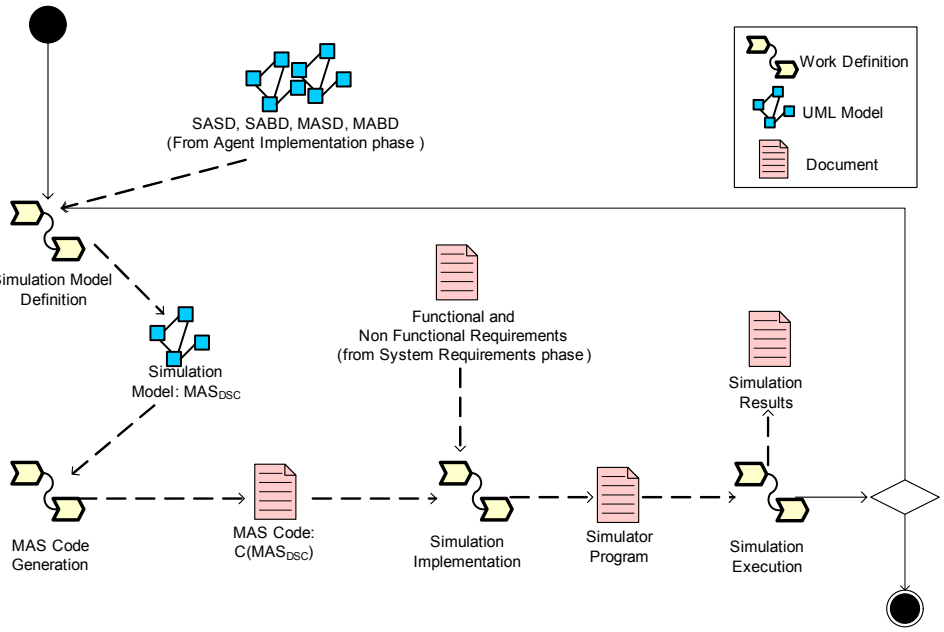
The *Simulation Model Definition* is enabled by the Distilled StateCharts (DSCs) formalism (Fortino *et al.*, 2004), which supports the specification of the behaviour of the agent types and the interaction protocols among the agent types of a MAS. DSCs were derived from Statecharts (Harel and Gery, 1997) and allow for the specification of the behaviour of Event-driven Lightweight Agents (ELAs), which are single-threaded entities capable of transparent migration and executing chains of atomic actions.

The DSC-based specification of a MAS, denoted as MAS_{DSC}, is expressed as:

$$MAS_{DSC} = \{Beh(AT_1), \ldots, Beh(AT_n)\},$$

where $Beh(AT_i) = <S_{Beh}(AT_i), E_{Beh}(AT_i)>$ is the DSC specification of the dynamic behaviour of the *i*-th agent type. In particular, $S_{Beh}(AT_i)$ is a hierarchical state machine incorporating the activity and the event handling of the *i*-th agent type, and $E_{Beh}(AT_i)$ is the related set of events to be handled triggering state transitions in $S_{Beh}(AT_i)$. In particular, $S_{Beh}(AT_i)$ is designed on the basis of a template compliant with the FIPA agent life cycle (FIPA, 2002) (see Figure 3). The Active Distilled StateChart (ADSC), inside the Active state, is to be refined by the agent designer. The deep history connector (H*) inside the Active state allows for agent migration based on a coarse-grained strong mobility model (Fortino *et al.*, 2004). In particular, the presence of the H* allows an agent to reenter in the state most recently exited by retaining the previous global state. The transition originating from the H* targets a state of the ADSC named Default History State (DHS).

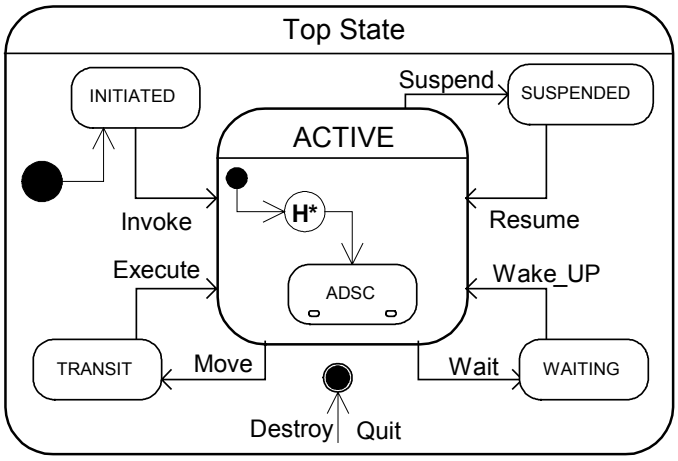
Figure 2 The DSC-based simulation method fragment



The *Simulation Model Definition* is an adapted fragment which takes as input the structural and dynamic diagrams (SASD, SABD, MASD and MABD diagrams) produced by the *Agent Implementation*, which are semiautomatically translated into a MAS_{DSC} as described in Section 3.

The *MAS Code Generation* is supported by the Mobile Active Object Framework (MAO Framework) (Fortino *et al.*, 2004), currently implemented in Java. Given the MAS_{DSC}, it produces $C(MAS_{DSC})$ representing the code of MAS_{DSC}. $Beh(AT_i)$ is translated into a composite object, which is the object-based representation of $S_{Beh}(AT_i)$, and into a set of related event objects of the MAOEvent type which represent $E_{Beh}(AT_i)$.

Figure 3 The FIPA-compliant DSC template



The *Simulation Implementation* and *Simulation Execution* are supported by MASSIMO (Multi-Agent System SIMulator framewOrk) (Fortino *et al.*, 2005a), a Java-based discrete-event simulation framework for MASs which allows for the validation and evaluation of:

- the dynamic behaviour (computations, communications and migrations) of individual and cooperating agents
- the basic mechanisms of the distributed architectures supporting agents, namely agent platforms
- the functionalities and emergent behaviours of applications and systems based on agents.

In particular the architecture of MASSIMO is composed of four basic layers:

1 *Low-level simulation framework*, which provides the basic classes (Agent, MetaAgent, Message and Timer) and the discrete-event simulation engine to program and simulate general-purpose agent-oriented systems

2 *Agent platform*, which is built atop the *low-level simulation framework* layer and provides two basic abstractions: the AgentServer, which represents the infrastructure where event-driven lightweight DSC-based agents (ELAs) run, and the VirtualNetwork, which represents a network of hosts on which AgentServers can be mapped. AgentServers interact with each other through signalling messages (MSG)

3 *ELA adapter*, which extends the Mobile Agent Adaptation Framework (MAAF) (Fortino *et al.*, 2004) and allows the mapping of ELAs, programmed through the MAO Framework, onto the *agent platform* layer

4 *User*, which makes available two abstract classes – UserAgent and UserAgentGenerator – which are extensions of Agent. UserAgent represents a user directly connected to an AgentServer who can create, launch and interact with ELAs. UserAgentGenerator models the generation process of UserAgents.

Moreover, the Start message allows for the activation of a UserAgent or a UserAgentGenerator, whereas the Reporting message, which targets a UserAgent, contains a report sent from an ELA owned by the UserAgent.

On the basis of functional and nonfunctional requirements and the MAS code, a simulator program can be implemented by using MASSIMO in the *Simulation Implementation*.

In the *Simulation Execution* the simulator program is executed to obtain the simulation results containing validation traces and performance parameter values. The validation of agent behaviours and interactions is carried out by automatically generated execution traces, whereas the performance evaluation relies on the specific MAS to be analysed; the performance evaluation parameters are therefore set *ad hoc*. The simulation results can be used to feed back the *Simulation Model Definition*.

2.4 Coding

The Coding phase is carried out by the *Code* (composed) method fragment selected from PASSI which produces the code of the MAS being developed. The *Code* is composed of two atomic method fragments:

1 *Code Reuse*, in which code generation is directly supported by the PTK. In particular, it is possible to generate not only the skeletons but also largely reusable parts of the methods implementation based on a repository of reused patterns and associated design descriptions. Currently, the pattern repository includes a set of reusable portions of JADE and FIPA-OS agents and corresponding behaviours; a more detailed description of the pattern repository can be found in Cossentino *et al.* (2003) and Chella *et al.* (2003).

2 *Code Refinement*, where code is manually completed by the programmer.

2.5 Deployment

The Deployment phase is carried out by the *Deployment* (composed) method fragment selected from PASSI, which specifies the distribution of the parts of the system (agents) across the available agent platforms. The *Deployment* is composed of only the *Deployment Configuration* atomic method fragment, which produces the deployment diagrams describing the allocation of agents to the available agent platforms and any constraints on agent migration. In particular, these diagrams also specify the libraries or hardware devices (sensors or actuators) that should be available on the agent platforms in order to ensure the proper system functionalities.

3 Adapting the design for the simulation

As introduced in Section 2, in order to simulate the MAS being developed, the work products of the *Agent Implementation* carried out in the Design phase must be translated into a Multi-Agent System Distilled StateChart Simulation Model (MAS_{DSC}), which represents the work product of the *Simulation Model Definition* (see Table 1). The input to the translation process consists of the SASD, SABD, MASD and MABD diagrams, whereas the output of the translation process is represented by a MAS_{DSC}.

The translation process is semiautomatic, which means that these diagrams are first automatically translated into a MAS_{DSC} skeleton, and then the MAS_{DSC} skeleton is manually refined through programming. In particular, the following steps are carried out:

Step 1 The agent types of the MAS_{DSC} are directly derived from the agent types of the MASD diagram through a one-to-one mapping.

Step 2 The interactions in terms of events exchanged between the agent types of the MAS_{DSC} are directly derived from the MABD diagram.

Step 3 The ADSC of an agent type is based on the SASD and the SABD diagrams of the agent type. As a SASD is a platform-dependent diagram (*e.g.*, FIPA-OS-based or JADE-based), the SASDs are designed to be DSC-oriented. In particular, attributes and methods of the agent type are inserted into the ADSC as state variables and supporting functions, respectively. These state variables and supporting functions need to be manually finalised, *i.e.*, the specific type of all the state variables is defined and the methods are implemented. The activities reported in the SABD diagram become states of the ADSC and the transitions among activities become transitions among the states corresponding to these activities. Finally, the ADSC is to be refined through manual programming, which is needed for model consistency and optimisation purposes. This refinement step involves the introduction/deletion of states, transitions, transition labels (event[guard]/action), state variables and supporting functions.

In the following we use a simple example to show how the semiautomatic translation from the work products of the *Agent Implementation* to a MAS_{DSC} can be obtained. The example MAS we considered is composed of two agent types:

1 an Information Retrieval Agent (IRA) whose task is to visit a given number of locations to retrieve information through a query

2 an Information Provider Agent (IPA) whose task is to process the query received from the IRA and to provide it with the query result.

The work products produced by the *Agent Implementation* activities are shown in Figures 4, 5, 6 and 7.

Figure 4 The MASD of the example MAS

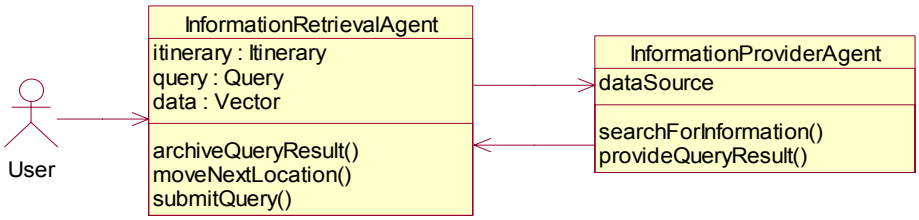


Figure 5 The MABD of the example MAS

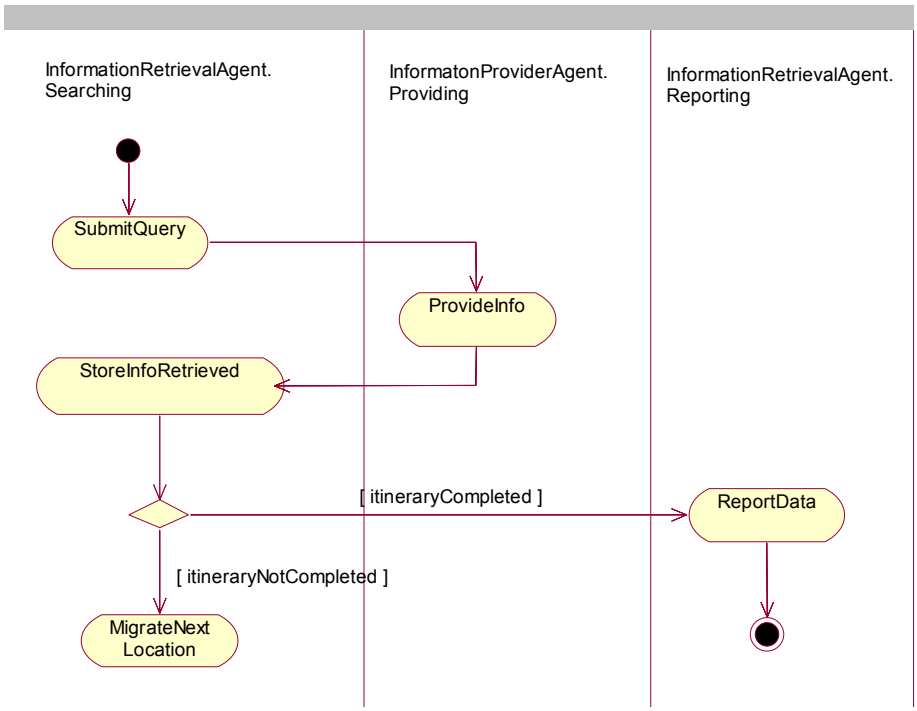
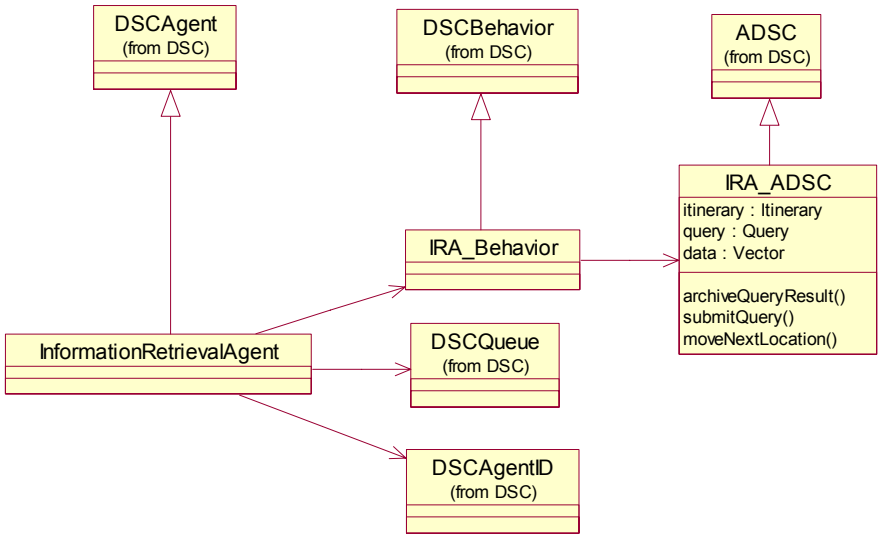
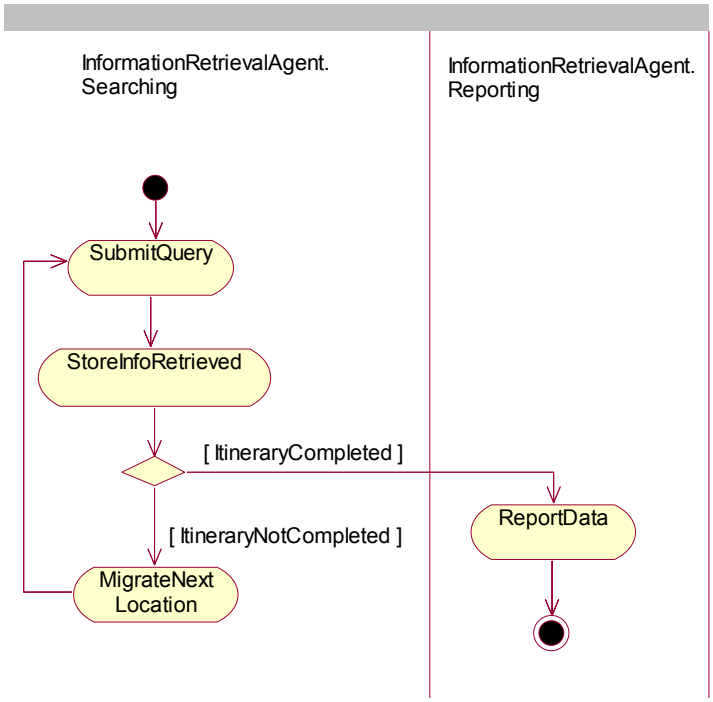


Figure 6 The (a) SASD and (b) SABD of the IRA



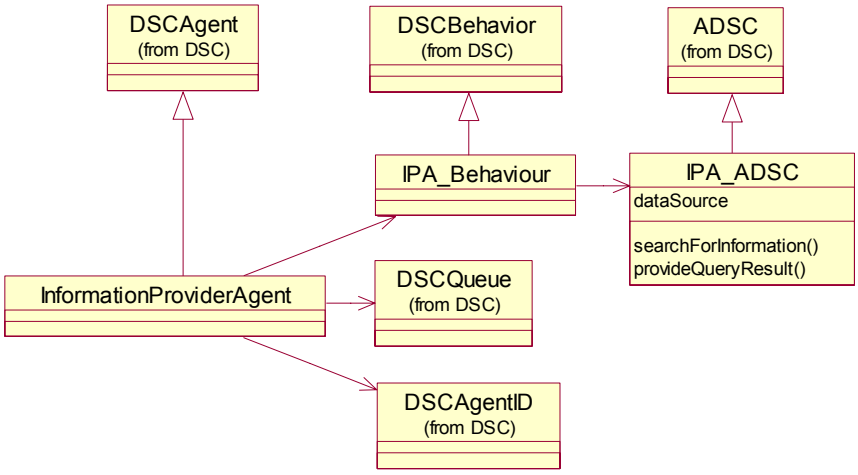
(a)

Figure 6 The (a) SASD and (b) SABD of the IRA (continued)



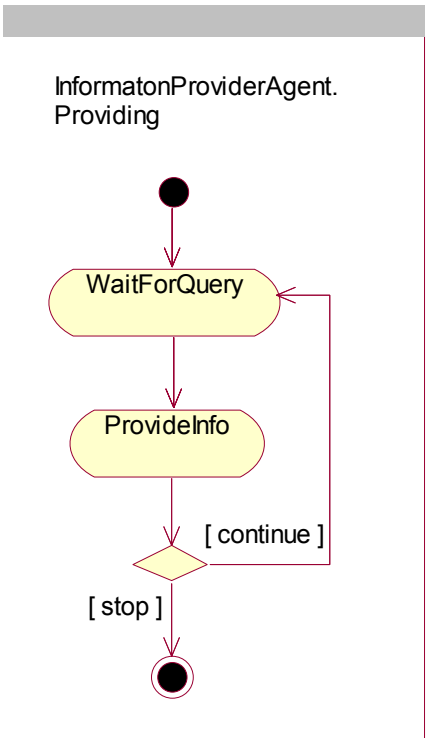
(b)

Figure 7 The (a) SASD and (b) SABD of the IPA



(a)

Figure 7 The (a) SASD and (b) SABD of the IPA (continued)



(b)

Given the MASD of the example MAS (Figure 4), the agent types of the MAS_{DSC} are INFORMATIONRETRIEVALAGENT and INFORMATIONPROVIDERAGENT.

Given the MABD of the example MAS (Figure 5), the events exchanged between the two agent types are: QUERYREQUEST(QUERY) and QUERYINFORM(QUERYRESULT), which correspond to the two main messages of the FIPA Query Protocol, which was selected for the communication between the two agents.

Given the SASD and SABD diagrams of the INFORMATIONRETRIEVALAGENT (see Figure 6), the ADSC skeleton of the INFORMATIONRETRIEVALAGENT of the MAS_{DSC} reported in Figure 8 was obtained. The names of the states of the ADSC have as suffix the names of the activities of the SABD diagram and as postfix 'Done', which means that the activity corresponding to the state has been carried out. The event labelling the transition from SubmitQueryDone to StoreInfoRetrieveDone corresponds to the message QueryInform sent from the IPA agent. The events labelling the transitions from StoreInfoRetrieveDone are derived from the guards of the selection block of the IRA SABD diagram (see Figure 6b). The event labelling the transition from MigrateNextLocationDone to SubmitQueryDone assumes the name of the activity corresponding to the target state as each transition of a DSC must be labelled by an event.

Figure 8 The ADSC skeleton of the IRA

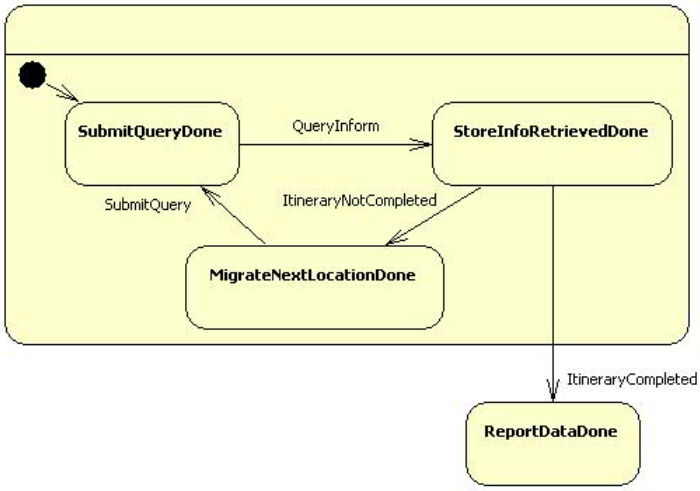
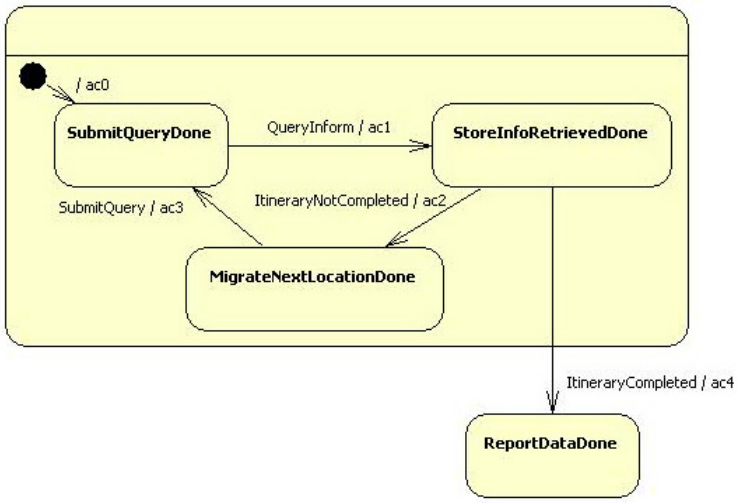


Figure 9 The refined ADSC of the IRA

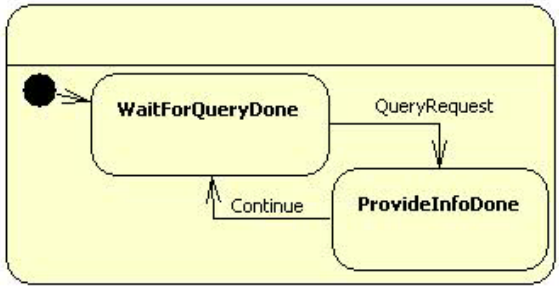



```
ac0: generate(new QueryRequest(self(), IPA, query));
ac1: QueryInform qi = (QueryInform)evt;
     archiveQueryResult(qi.getInfo());
     if (itinerary.hasNextLocation())
       generate(new ItineraryNotCompleted(self()));
     else
       generate(new ItineraryCompleted(self()));
ac2: Location nextLoc = itinerary.getNextLocation();
     generate(new Move(self(), bextLoc));
     generate(new SubmitQuery(self()));
ac3: reportData();
ac4: ac0;
```

The ADSC of the INFORMATIONRETRIEVALAGENT which was obtained after refinement is shown in Figure 9. The actions have been purposely defined 'by programming' on the basis of the state variables and supporting functions derived from the SASD diagram (see Figure 6a).

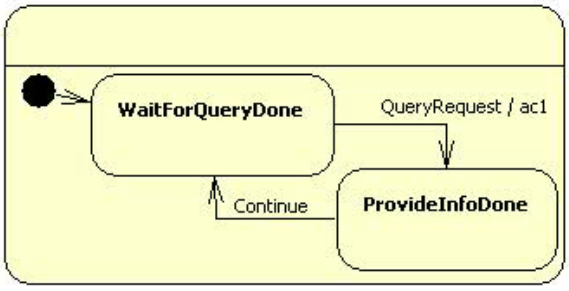
Given the SASD and SABD diagrams of the INFORMATIONPROVIDERAGENT (see Figure 7), the ADSC skeleton of the INFORMATIONPROVIDERAGENT of the MAS_{DSC} was obtained (see Figure 10). Two states are derived: WaitForQueryDone, referring to the end of the WaitForQuery activity, and ProvideInfoDone, referring to the end of the ProvideInfo activity. The event labelling the transition from WaitForQueryDone to ProvideInfoDone corresponds to the message QueryRequest sent from the IRA agent. The Continue event labelling the transition from ProvideInfoDone to WaitForQueryDone is derived from the guard of the selection block of the IPA SABD diagram (see Figure 7b).

Figure 10 The ADSC skeleton of the IPA



The ADSC of the INFORMATIONPROVIDERAGENT which was obtained after refinement is shown in Figure 11.

Figure 11 The refined ADSC of the IPA




```
ac1: QueryRequest qr=(QueryRequest)evt;
     Result r = searchForInformation(qr.getQuery());
     generate(new QueryInform(self(), qr.getSource(), r));
     generate(new Continue(self(), self()));
```

4 A case study: from the analysis to the validation of an agent-based e-marketplace

An electronic marketplace (e-marketplace) is a platform for buyers and sellers exchanging products and services (Feldman, 2000; Ripper *et al.*, 2000):

- Buyers specify the items they want to buy, along with their desired price ranges.
- The e-marketplace then matches trading partners for the buyers and provides the Request for Quotation (RFQ).
- On the basis of the specification and price range, sellers return the quotation to the buyers and wait for the confirmation.
- After receiving all quotations, buyers can select the best offer and issue a purchase order to the selected sellers.

Nowadays, many e-marketplaces are based on software agents which are capable of fully supporting and automating the stages of the Consumer-Buying Behaviour (CBB) model (Guttman *et al.*, 1998; Maes *et al.*, 1999). The CBB model defines the decision process which consumers undergo when purchasing a product. Such a process is articulated in six stages:

Stage 1 *Need identification*: This stage characterises the buyer, who becomes aware of some unmet/desired need. Within this stage, the buyer can be stimulated through product information.

Stage 2 *Product brokering*: This stage comprises the retrieval of information to help determine what to buy. This encompasses the evaluation of product alternatives based on buyer-provided criteria. The result of this stage is the 'consideration set' of products.

Stage 3 *Merchant brokering*: This stage combines the 'consideration set' from the previous stage with merchant-specific information to help determine who to buy from. This includes the evaluation of merchant alternatives based on buyer-provided criteria (*e.g.*, price, warranty, availability, delivery time, reputation).

Stage 4 *Negotiation*: This stage is about how to settle on the terms of the transaction. The negotiation varies in duration and complexity depending on the market.

Stage 5 *Purchase and delivery*: The purchase and delivery of a product can either signal the termination of the negotiation stage or occur sometime afterwards.

Stage 6 *Product service and evaluation*: This postpurchase stage involves product service, customer service and an evaluation of the satisfaction of the overall buying experience and decision.

The objective of our case study is to apply PASSIM to the design and validation of an AeM which supports Stages 3, 4, and 5 through the following specific consumer-buying process (Fortino *et al.*, 2005b):

- *Request Input*. When users wish to buy a product, they identify a set of product parameters (product description, maximum price P_{MAX}), log into the e-marketplace and submit a request containing the product parameters. The e-marketplace checks if users are trustworthy (*i.e.*, from a commercial and security viewpoint) and decides if requests can be accepted. If so, the Consumer Assistant System (CAS) of the e-marketplace starts satisfying the user request.
- *Searching*. The CAS obtains a list of locations of vendors by using the Yellow Pages Service (YPS) of the e-marketplace. The YPS is a federation of autonomous components at which vendors register to advertise their products. In particular the following YPS organisations were established:
  - a Centralised: Each YPS component stores a complete list of vendors.
  - b One-Neighbor Federated: Each YPS component stores a list of vendors and keeps a reference to only one other YPS component.
  - c M-Neighbors Federated: Each YPS component stores a list of vendors and keeps a list of at most M YPS components.
- *Contracting and evaluation*. The CAS interacts with the vendors found to request an offer (P_{OFFER}) for the desired product, evaluates those received, and selects an offer for which the price is acceptable (*i.e.*, $P_{OFFER} \leq P_{MAX}$), if any.
- *Payment*. The CAS purchases the desired product from the selected vendor using a given amount of e-cash (or bills). The following steps are performed to execute the money transaction between the CAS and the vendor:
  - Step 1 The CAS gives the bills to the vendor.
  - Step 2 The vendor sends the bills to its bank.
  - Step 3 The bank validates the authenticity of the bills, exempts them from reuse, and, finally, issues to the vendor an amount of bills equal to that previously received.
  - Step 4 The vendor notifies the CAS.
- *Reporting*. The CAS reports the buying result to the user.

This description can be considered an initial requirements document on the basis of which the Requirements Specification phase is carried out. In the following subsections selected work products of the first four phases of PASSIM (see Section 2) are shown and described. In particular, Section 4.1 presents the Requirements Specification work products, Section 4.2 shows the Design work products, and, finally, Section 4.3 shows the establishment and the results of the Simulation phase, which allow for both functional validation and performance evaluation of the MAS being developed.

4.1 The Requirements Specification phase

From the previously reported description of the system to be designed, the AId (see Section 2.1) was drawn, which reports three actors (User, Vendor and Bank) and the use cases coming from the *Domain Description*, which were packaged into the following six agents:

1 *User Assistant Agent (UAA)* is associated with a user and assists her/him in looking for a specific product that meets her/his needs and in buying the product according to a specific buying policy.

2 *Yellow Pages Agent (YPA)* represents an entry point of the federated yellow pages service (or 'Yellow Pages') which provides the location of agents selling a given product.

3 *Vendor Agent (VA)* represents the vendor of specific goods.

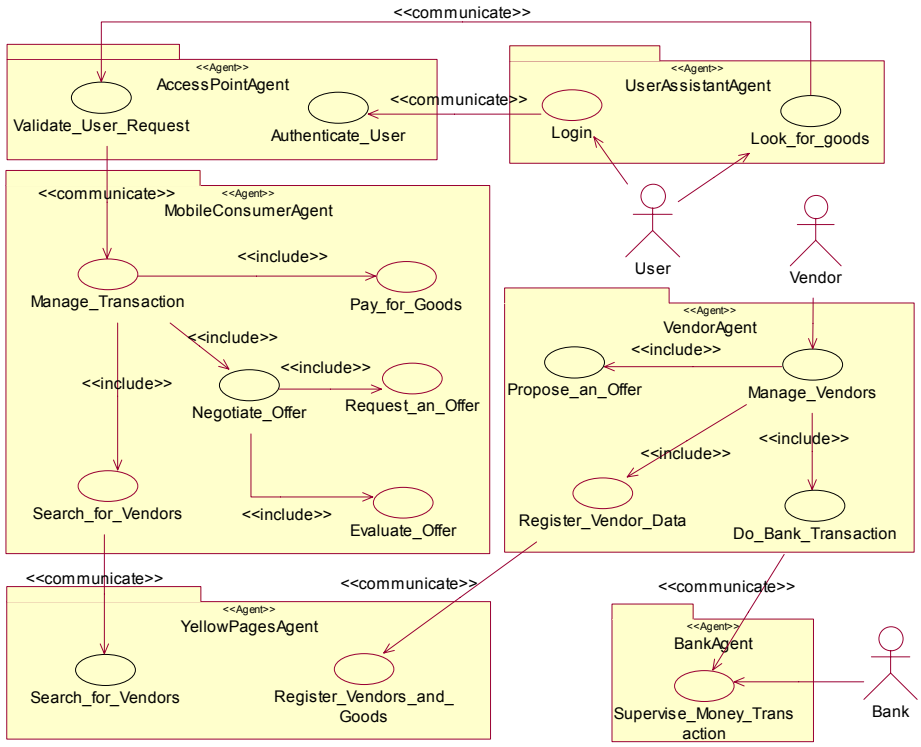
4 *Mobile Consumer Agent (MCA)* is an autonomous mobile agent dealing with searching, contracting, evaluation and payment of goods.

5 *Access Point Agent (APA)* represents the entry point for the e-marketplace, accepts requests for buying a product from a registered UAA and fulfils them by generating a specific MCA.

6 *Bank Agent (BA)* represents a reference bank of MCA and VA.

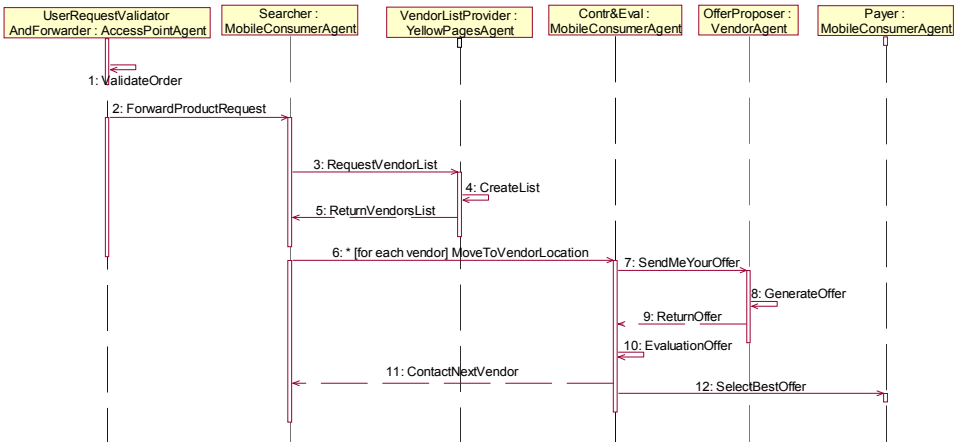
It is worth noting that the <<communicate>> relationship shown in Figure 12 represents agent interactions.

Figure 12 The AId for the proposed case study



On the basis of the AId, the Roles Identification diagram (RId) was designed. A portion of the obtained RId is shown in Figure 13 where the APA (*UserRequestValidatorAndForwarder* role), after validating the order, forwards it to the MCA (*Searcher* role); afterwards the MCA asks the YPA (*VendorListProvider* role) for the vendors list. After getting the list, the MCA (*Contr&Eval* role) contacts all the VAs (*OfferProposer* role) and asks them for their offers. Finally, the MCA selects the best offer and pays for the product (*Payer* role).

Figure 13 A portion of the RId regarding a search scenario



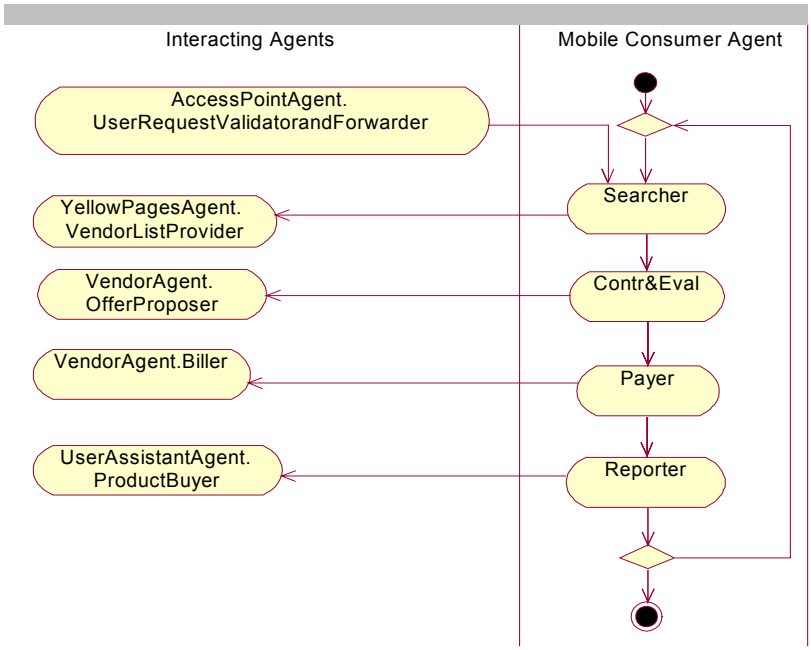
An initial definition of the dynamic behaviour of each agent is the work product produced by the last atomic method fragment of this phase (*Tasks Specification*). The *Tasks Specification* produces a set of Tasks Specification diagrams (one for each identified agent), which are UML activity diagrams representing the agent tasks. Each diagram is composed of two swim-lanes (see Figure 14): the right-hand side highlights the roles of the agent which the diagram refers to and the activities the agent performs in playing these roles, whereas the left-hand side reports the roles played by other agents interacting with the agent on the right-hand side.

In Figure 14, the Tasks Specification diagram of the MCA is shown. In particular, the MCA is involved in:

- searching the list of vendors through a query to the YPA (*Searcher* role)
- contracting with VAs and evaluating their offers (*Contr&Eval* role)
- buying the product from the VA proposing the best offer (*Payer* role)
- reporting the transaction results to the UAA (*Reporter* role).

Afterwards the MCA can either play the *Searcher* role again or be terminated.

Figure 14 The tasks specification diagram for the MCA



4.2 *The Design phase*

The *Agent Society* method fragment (see Section 2.2.1) produces diagrams which represent social interactions and dependencies among the identified agents (see Section 4.1). A portion of the DOD diagram is reported in Figure 15 in which some concepts, predicates and actions used to define the problem domain are shown. For instance, the *Vendor* concept (representing the vendor of the real-world scenario) is related with the *Product*(s) it sells. A vendor registers its products in the agent-based yellow pages service by executing the *RegisterProduct* action, which is performed by the VA (action Actor), and its outcome is received by the YPA (action Receiver).

A portion of the COD diagram is reported in Figure 16. It shows three identified agents (APA, VA, MCA) and two communications among them (Forward_Product_Request, Offer_Request). In particular, the Offer_Request communication happens when the MCA asks the VA for the best offer (see the scenario reported in Figure 13). This communication refers to the OfferPrice predicate from the ontology of Figure 15 and adopts the FIPAQuery agent interaction protocol and the RDF content language. Roles played by agents during the interaction (as described in the RIds) are reported at the beginning and the end of the association line.

The *Agent Implementation* method fragment (see Section 2.2.2) produces work products representing the MAS architecture. In particular, a portion of the MASD diagram, which describes the structure of the VA, MCA and APA agents, is shown in Figure 17. It is worth noting that the VA is in relationship with a human actor this is an extension of UML that is useful for representing all the possible agent relationships (communications and GUI-based interactions with the user) in a unique diagram.

Figure 15 A portion of the DOD diagram

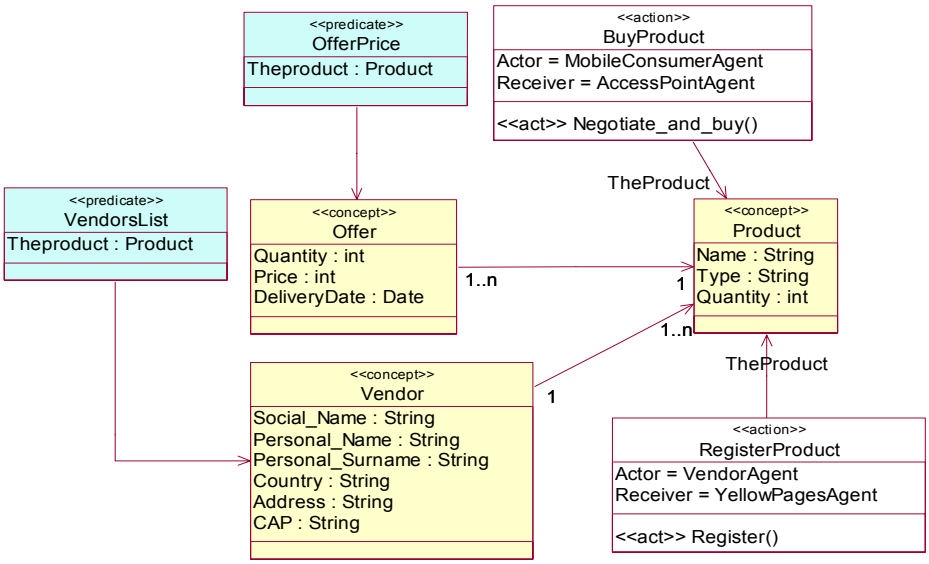


Figure 16 A portion of the COD diagram

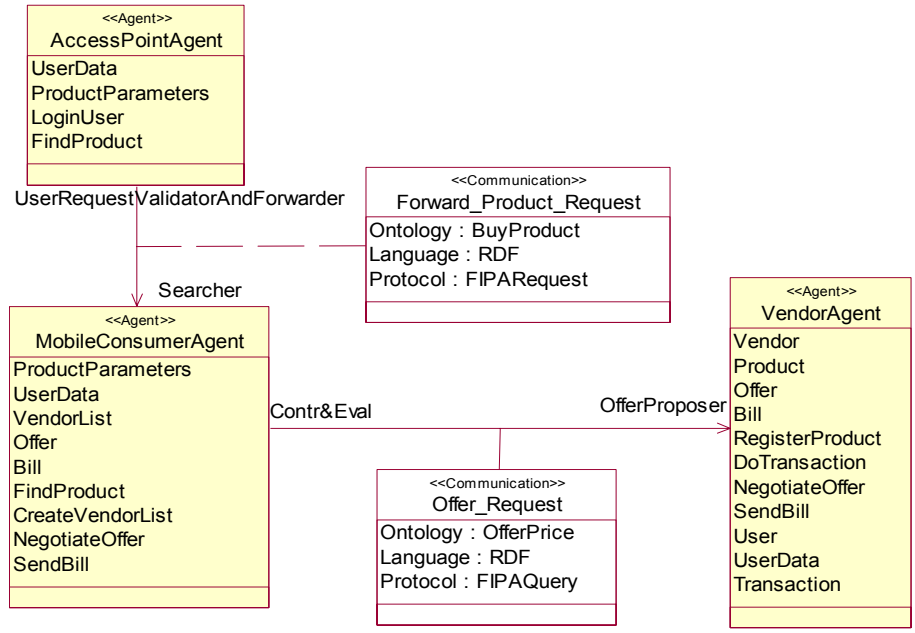
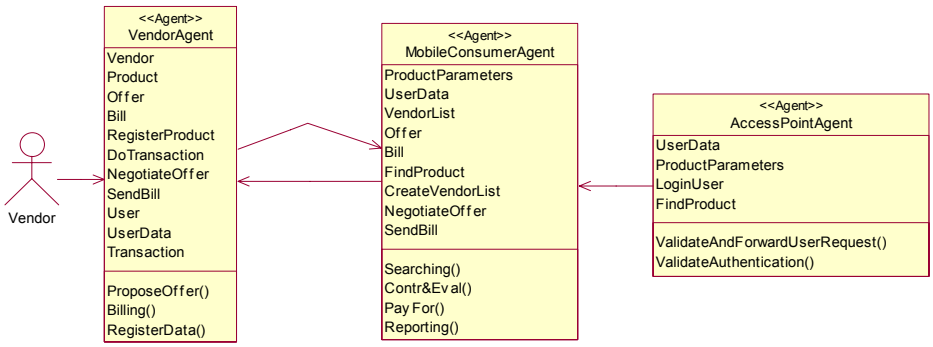


Figure 17 A portion of the MASD diagram



A portion of the obtained MABD diagram is reported in Figure 18, which illustrates the activities occurring during the Vendor_Request communication between MCA and YPA and the Offer_Request communication between MCA and VA. In particular, this portion of the MABD diagram describes the request for the VA list from the MCA to the YPA, the migration of the MCA to the retrieved VA location and the contracting phase carried out by the MCA with the VA.

Figure 18 A portion of the MABD diagram with some interactions among MCA, YPA and VA

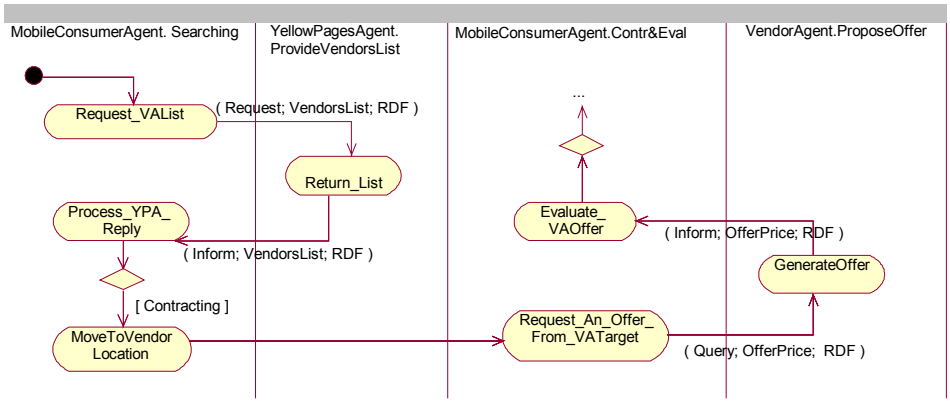


Figure 19 shows the SABD diagram for the MCA, which provides a high-level specification of the behaviour of the MCA. In particular, the MCA plays four different roles in the following sequence: Searching, Contr&Eval, PayFor and Reporting. They also correspond to the phases of the MCA life cycle. In particular, in the Searching phase, the MCA moves to the location of the next YPA (YPATarget), requests the list of vendors (VAList) and processes the reply (YPA_Reply). If the Searching phase is not completed ([Searching] is evaluated to be true), the MCA continues searching. If the guard [Contracting] holds (*i.e.*, the VAList is not empty), the MCA passes into the Contr&Eval phase. If the guard [Reporting] holds (*i.e.*, the VAList is empty), the MCA directly goes into the Reporting phase. In the Contr&Eval phase, the MCA moves to the location of a vendor in the VAList (VATarget), requests an offer (VAOffer) and

evaluates it. If the MCA decides to accept the received VAOffer (*i.e.*, the guard [BuyingSoon] holds) or another received VAOffer (*i.e.*, the guard [MovingAndBuying] holds), it passes into the PayFor phase. If the MCA desires a new offer, it keeps contracting (*i.e.*, guard [Contracting] holds true). If no offer is selected the MCA goes into the Reporting phase (*i.e.*, guard [Reporting] holds true). Finally, in the Reporting phase, the MCA moves to the APA location and reports to its UAA.

Figure 19 The SABD diagram for the MCA

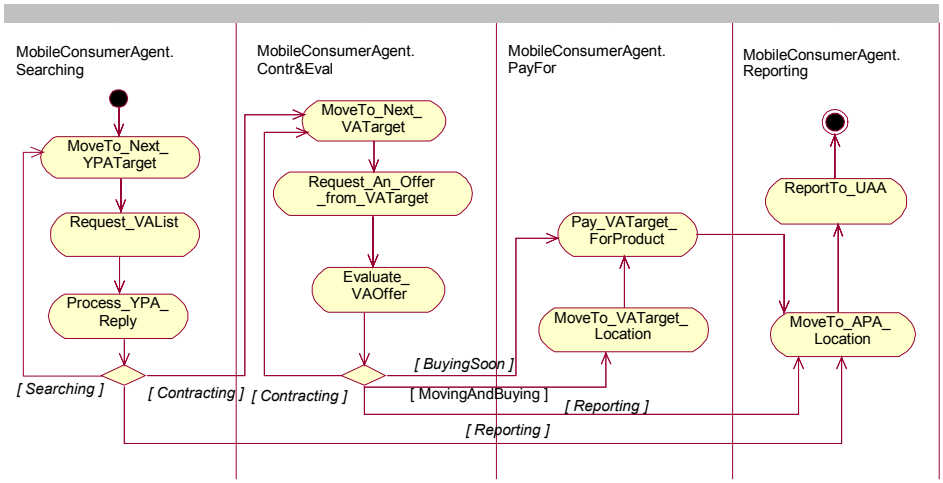
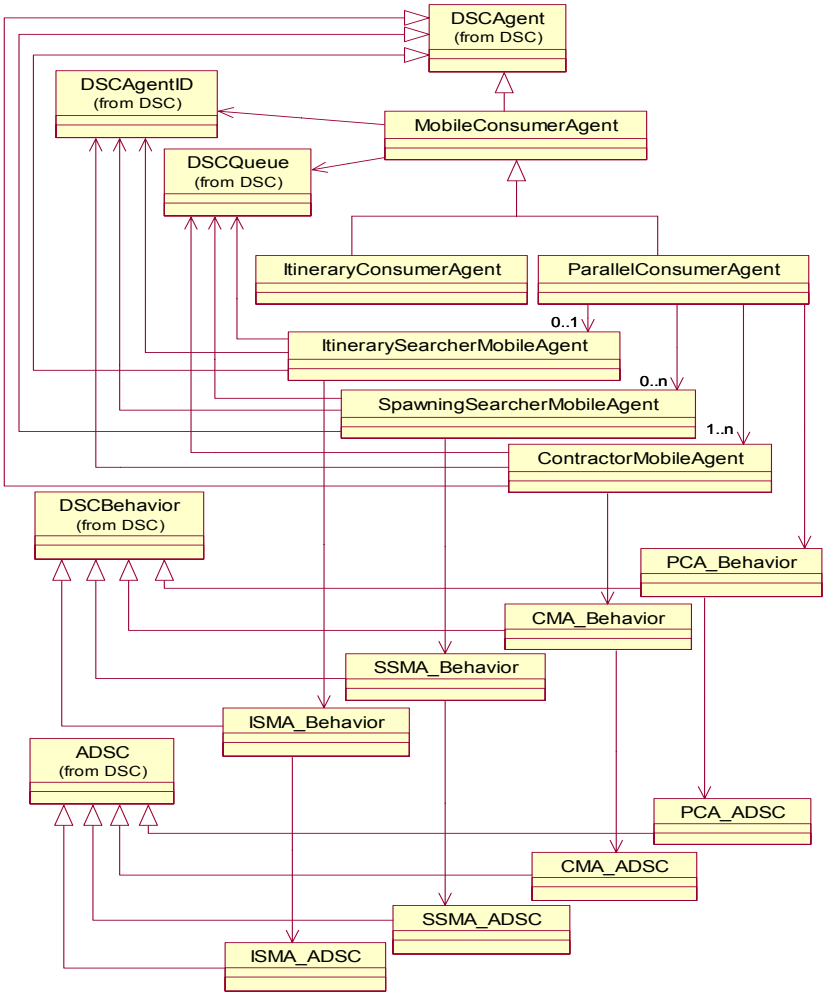


Figure 20 shows the SASD diagram for the MCA and its derived agents. In particular, two specific MCAs are derived:

1 the ItineraryConsumerAgent (or ICA), which performs the Searching and Contr&Eval phases (see Figure 19) by sequentially moving from one location to another within the e-marketplace

2 the ParallelConsumerAgent (or PCA), which performs the Searching and Contr&Eval phases (see Figure 19) by means of the support of a set of mobile agents: the ItinerarySearcherMobileAgent or the SpawningSearcherMobileAgent for carrying out sequential or parallel searching of vendors during the Searching phase; and the ContractorMobileAgent for carrying out parallel negotiation during the Contr&Eval phase.

Figure 20 The SASD diagram for the MCA and derived agents



4.3 The Simulation phase

The aim of the Simulation phase is the functional validation of the designed AeM and the performance evaluation of different types of MCAs for optimisation purposes. In particular, the functional validation is carried out on the basis of simple simulation scenarios aiming at validating the behaviour of the agent types, the agent interaction protocols and the global behaviour of the AeM. The performance evaluation is carried out to evaluate the completion time of the buying task of different types of MCAs.

In the following subsections, first the refined DSC-based MCAs derived from the *Simulation Model Definition* are described (Section 4.3.1); then, the functional validation (Section 4.3.2) and the performance evaluation (Section 4.3.3) are presented.

4.3.1 DSC-based MCAs

Two types of DSC-based MCAs were obtained according to the adapting *Simulation Model Definition* (see Section 3): ICA and PCA. Both ICA and PCA are equipped with policies for searching and buying (see Table 2) during the *Searching* and the *Contr&Eval* phases, respectively.

Table 2 Searching and buying policies of MCA

Searching Policy (SP)	*Description*
ALL	All YPA agents are contacted.
PA-PARTIAL	A subset of YPA agents are contacted.
OS-ONE-SHOT	Only one YPA agent is contacted.
Buying Policy (BP)	*Description*
MP-Minimum Price	The MCA first interacts with all the VA agents; then, it buys the product from the VA offering the best acceptable price.
FS-First Shot	The MCA interacts with the VA agents until it obtains an offer for the product at an acceptable price, then it buys the product.
FT-Fixed Trials	The MCA interacts with a given number of VA agents and buys the product from the VA which offers the best acceptable price.
RT-Random Trials	The MCA interacts with a random number of VA agents and buys the product from the VA which offers the best acceptable price.

In the following, we focus on the PCA, as the ICA possesses a more simple behaviour, which is encompassed by the PCA. Figure 21 shows the refined ADSC of the PCA, which was derived from the MASD, MABD, SASD and SABD diagrams (see Figures 17–20) of the MCA and from the SABD diagram specific to the PCA (not reported here for the sake of brevity), which is a specialisation of the SABD diagram of the MCA.

The messages that the MCA exchanges with the YPA, VA and UAA during its life cycle, reported in the MABD diagram, are implemented through external events in the ADSC; the association between messages and events is reported in Table 3 for the interactions with YPA and VA.

The names of the composite states of the ADSC correspond to the names of the phases of the MCA shown in the related SABD diagram (see Figure 19). For the sake of modularity the Searching and Contr&Eval states are embodied in the Search&Buy state.

The activities reported in the SABD diagram are implemented by the action chains of the ADSC; the association between activities of the SABD diagram and action chains is reported in Table 4.

Figure 21 The ADSC of the PCA

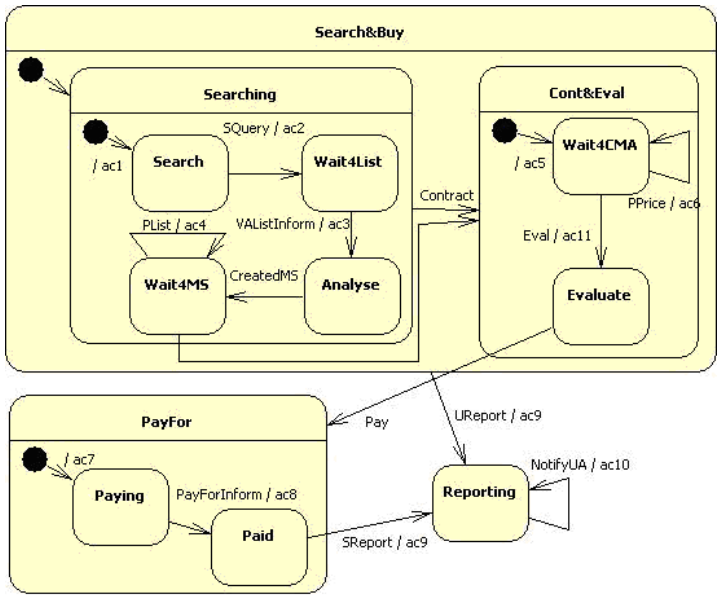


Table 3 Association between the messages of the MABD diagram of the MCA and the ADSC events of the PCA

MABD message	*Sender → Receiver*	*ADSC event*
(Request, VendorsList, RDF)	MCA → YPA	VAListRequest
(Inform, VendorsList, RDF)	VA → YPA	VAListInform
(Query, OfferPrice, RDF)	MCA → VA	OfferPriceQuery
(Inform, OfferPrice, RDF)	VA → MCA	OfferPriceInform
(Request, Payment, RDF)	MCA → VA	PayForRequest
(Inform, Payment, RDF)	VA → MCA	PayForInform

Table 4 Association between the activities of the SABD diagram of the MCA and the ADSC action chains of the PCA

SABD activity	*ADSC action chain*
MoveTo_Next_YPATarget	ac1, ac2
Request_VAList	ac3
Process_YPA_Reply	ac4
MoveTo_Next_VATarget	sa1
Request_An_Offer_From_VA_Target	ac5
Evaluate_VAOffer	ac6
MoveTo_VATarget_Location	ac11
Pay_VATarget_ForProduct	ac7, ac8
MoveTo_APA_Location	ac9
ReportTo_UAA	ac10

The PCA fulfils the searching phase in the Searching state. In particular, as soon as the PCA is created, it moves (*ac1*) to the first YPA location and locally interacts (*ac2*) with the *YPATarget* by sending it the *VAListRequest* event. The *YPATarget* replies to the PCA with the *VAListInform* event, which can contain a list of VAs with the linked YPAs. After processing the reply (*ac3*), the PCA can do one of the following:

- Create an *Itinerary Searcher Mobile Agent* (ISMA), which sequentially moves from one YPA location to another, if the YPS organisation is of the One-Neighbor Federated type, and pass (*ac4*) into the contracting phase as soon as a *PList* event sent by the ISMA is received.
- Create M *Spawning Searcher Mobile Agents* (SSMAs), if the YPS organisation is of the M-Neighbors Federated type, and pass (*ac4*) into the contracting phase when all the *PList* events sent by the directly created SSMAs are processed. In particular, an SSMA moves to the assigned YPA and, in turn, creates a child SSMA for each reachable YPA. This parallel searching technique generates a spawning tree, with SSMAs as nodes, which is rooted in the PCA. If an SSMA interacts with a YPA which has already been visited by an SSMA belonging to the same spawning tree, the YPA notifies the SSMA, which then returns to its parent.
- Directly pass into the contracting phase if the YPS organisation is of the Centralised type.
- Report an unsuccessful search to the UAA.

The contracting phase accomplished in the Contr&Eval state involves the creation (*ac5*) of a Contractor Mobile Agent (CMA) according to the modes reported below. Each CMA is able to move to the assigned VA location, contract with the VA, and report the obtained offer. The VA offers (*PPrice* events) reported by the CMAs are evaluated and a decision about when and from which VA to purchase is therefore taken (*ac6*). In the PayFor state the PCA pays (*ac7*) the VA using the *PayForRequest* event, which contains the bills. After receiving the *PayForInform* event, the PCA passes (*ac8*) to the Reporting state from where it moves back (*ac9*) to the original APA location and finally reports (*ac10*) to its UAA.

When using agent techniques in e-marketplaces, a large number of agents are generated in the e-marketplace network, which could lead to many problems such as server loading, network congestion and, more generally, scalability of the whole system (Leung *et al.*, 2004). So to optimise the performance of the PCA during the *Contr&Eval* phase with respect to time and resources, two types of CMAs (see Figure 22) have been defined:

1 *Full Parallel CMA (CMA_FP)*: The PCA spawns an instance of the CMA_FP for each VA location so that the CMA_FP contracts with the assigned VA and returns the obtained offer to the PCA. The advantage of this solution is that CMAs, once created by the PCA, can soon move to the assigned VA location and contract with the VA, thus minimising waiting times. However, the creation of a large number of CMAs on a single agent server can increase the agent server load as well as the network congestion in the proximity of the agent server. Moreover, if the buying policy is of the MP type, such a solution is effective; otherwise, such a solution would create more CMAs than needed.

2 *Binary CMA (CMA_BIN)*: After organising the list of the VAs retrieved in the *Search* phase as a binary tree, the PCA spawns a CMA_BIN to the VA location, the root of the tree. A CMA_BIN, in turn, spawns at most two other CMA_BIN agents if the left and/or right branches/leaves exist. In this operational mode, at most two agents are created on a single-agent server, reducing the server load due to agent creation and the network congestion due to agent migration (Wang *et al.*, 2002). According to the way the CMA_BIN returns the results of the negotiation with the VA to the PCA, the following types of CMA_BIN have been modelled:

   - *CMA_BIN_FW_R2PCA*: The agent directly reports to the PCA through an external event. The advantage of this solution rests on its simplicity, whereas if the number of CMAs created is high, there would be a high number of external events targeting the PCA, which would become a bottleneck.
   - *CMA_BIN_FW_R2O*: The agent reports to its owner (*i.e.*, the CMA agent that has spawned it) through an external event. In this way, only the root CMA reports to the PCA. In this mode, the disadvantage of the previous solution is avoided.
   - *CMA_BIN_BW_R2O*: The agent reports to its owner (*i.e.*, the agent that has spawned it) by moving to its site. Also in this case, only the root CMA reports to the PCA. This operational mode preserves the same advantage as the previous one and, in addition, can be effectively exploited in case the agents can only communicate through local interactions (*e.g.*, based on tuple-based systems).

Figure 23 shows the ADSC of the *CMA_BIN_FW_R2PCA*; the ADSCs of the other CMA_BINs are variants of the ADSC of the *CMA_BIN_FW_ R2PCA*. Migration and child spawning are carried out in the Migrate_And_Create state, whereas negotiation is carried out in the Contract state. In particular, after its creation the CMA moves to the location of the assigned VA (ac0), where it tries to spawn two other CMAs, and goes into the Contract state (ac1). In this state, the CMA sends the OfferPriceRequest event to the VA (ac2), processes (ac3) the offer contained in the OfferPriceInform event and, finally, reports to the PCA (ac4).

Figure 22 The SASD of the CMA

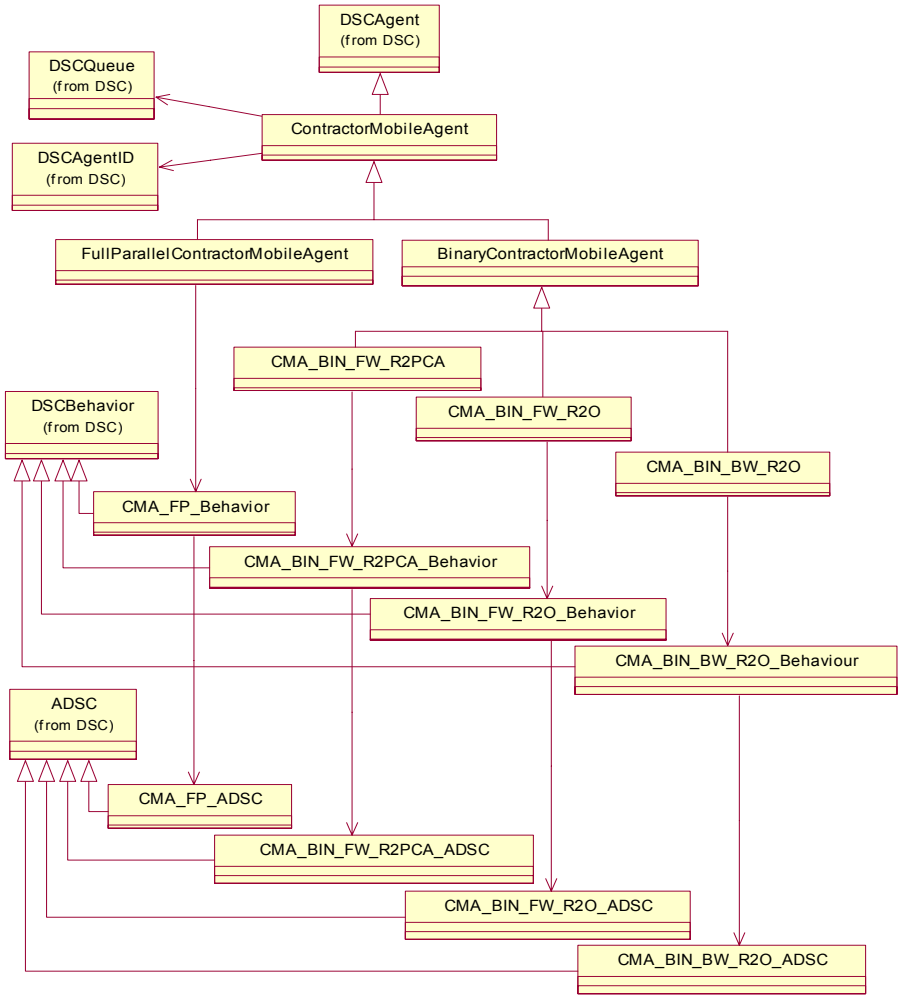
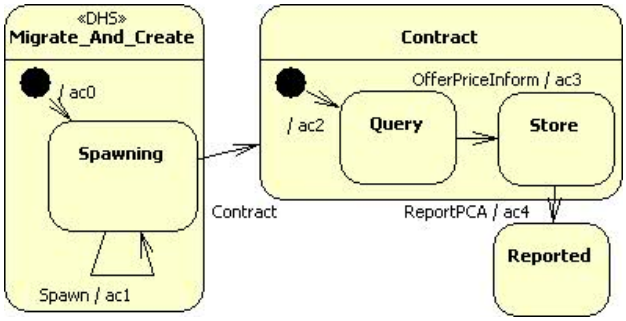


Figure 23 The ADSC of the CMA_BIN_FW_R2PCA



4.3.2 Functional validation

Functional validation is supported by MASSIMO through the generation of event traces, which can be analysed off-line to validate agent behaviours, agent interaction protocols and the behaviour of the whole MAS.

Validation of a single agent type behaviour relies on a simple simulation scenario, which allows for the generation of the response of the agent behaviour to all its admissible events. Validation of agent interaction protocols is based on a simple simulation scenario, which allows for the generation of the flow of events exchanged between the involved agents. Validation of the whole system is carried out by setting more complex simulation scenarios. In particular, the simulation scenario for the validation of the global behaviour of the AeM, also used during the performance evaluation phase, was set up as follows:

- Each stationary agent (UAA, APA, YPA, VA, BA) executes in a different agent server.
- Agent servers are mapped onto different network nodes, which are completely connected through links having the same characteristics and modelling the communication delay (δ) as a log-normally distributed random variable.
- Each VA is reachable from any YPA and sells the same set of products.
- Each product is always offered by a VA at a fixed price, which is an integer number uniformly distributed between a minimum (PP_{MIN}) and a maximum (PP_{MAX}).
- The user is willing to pay, for a desired product, a maximum price P_{MAX}, which is an integer value between PP_{MIN} and PP_{MAX}.

An indirect functional validation of the AeM was carried out by defining the following index, calculating it through both analytical methods and simulation, and comparing the results:

- the *Probability of Successful Buy* (P_{SB}), which is defined as the probability of successfully buying a desired product within the e-marketplace.

On the basis of the assumptions made for the simulated e-marketplace, P_{SB} can be easily calculated as follows:

$$P_{SB} = 1 - [(PP_{MAX} - P_{MAX})/(PP_{MAX} - P_{MIN} + 1)]^{V},$$

where:

V = the number of VAs contacted by the MCA for buying the product

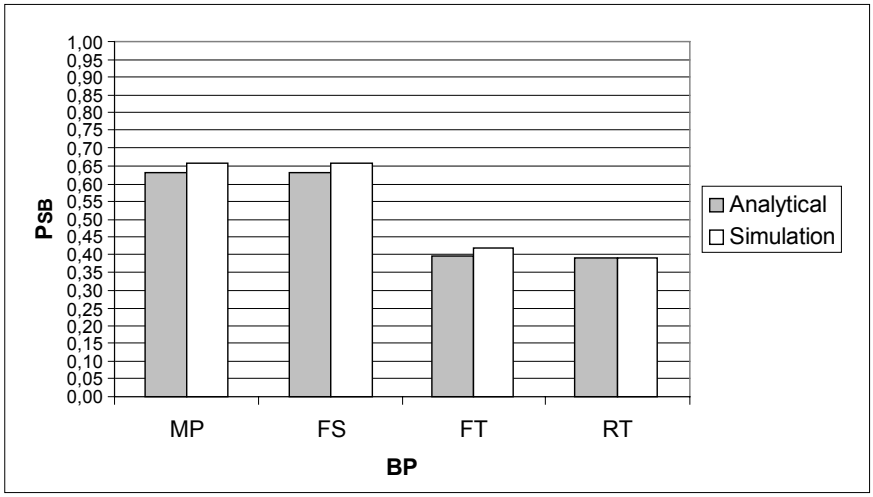
$PP_{MAX} - P_{MAX}$ = represents the number of prices that exceed P_{MAX} (*i.e.*, that are not acceptable to the user)

$PP_{MAX} - PP_{MIN} + 1$ = represents the number of all the possible prices for the product.

V depends on the BP adopted by the MCA; in particular, if BP is of the MP type or of the FS type, $V = N_{VA}$; if BP is of the FT type, V is $V_{FT} = N_{VA}/2 + 1$, as in the simulations the MCA always performs $N_{VA}/2 + 1$ trials; if BP is of the RT type, V belongs to the range $[1..N_{VA}]$.

The values of P_{SB} calculated both analytically and through simulation for each defined BP and with $PP_{MAX} = 200$, $PP_{MIN} = 100$, $P_{MAX} = PP_{MIN}$ and $N_{VA} = 100$ are reported in Figure 24. It is worth noting that the analytical value for BP = RT is calculated by using the mean value of the uniform distribution defined in the range $[1..N_{VA}]$.

Figure 24 Evaluation of P_{SB} for the defined BPs with $PP_{MAX} = 200$, $PP_{MIN} = 100$, $P_{MAX} = PP_{MIN}$ and $N_{VA} = 100$



Such results confirm that the global behaviour of the AeM is correct and this confirmation also provides an indirect functional validation of the AeM.

4.3.3 Performance evaluation

The aim of the performance evaluation phase is to evaluate and compare the efficiency of the five types of MCA (ICA, PCA/CMA_FP, PCA/CMA_BIN_FW_R2PCA, PCA/CMA_BIN_FW_R2O, PCA/CMA_BIN_BW_R2O) in terms of the following performance index:

Buy Task Completion Time (T_{BTC}) = $T_{CREATION} - T_{REPORT}$

where $T_{CREATION}$ is the creation time of the MCA and T_{REPORT} is the reception time of the MCA report.

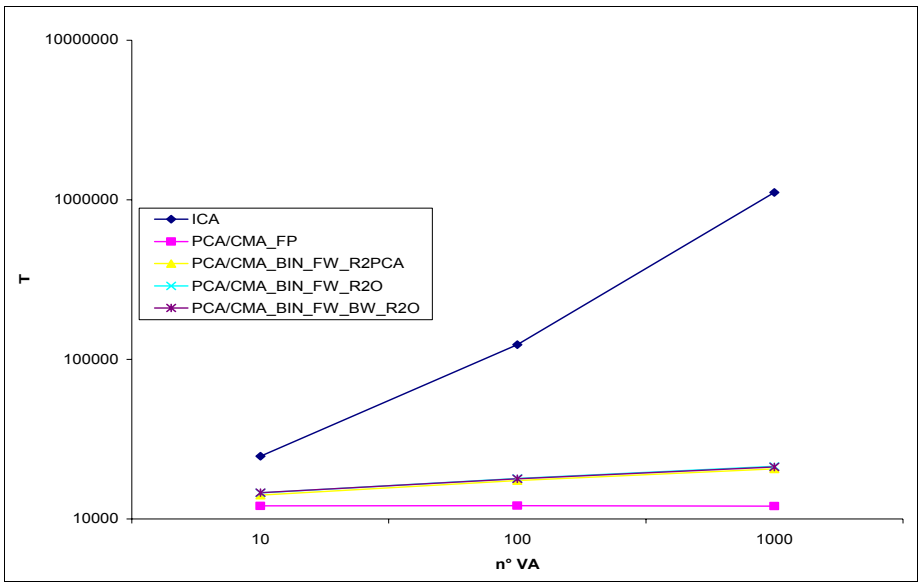
Given the scenario described in Section 4.3.2, the evaluation of the T_{BTC} performance index is focused on an MCA adopting a Searching Policy (SP) of the ALL type and a Buying Policy (BP) of the MP type (see Table 2); moreover it is supposed that $P_{MAX} = PP_{MAX}$, thus always guaranteeing a successful purchase at the best price.

The results, obtained by adopting a YPA organisation in which the YPAs are logically connected as a binary tree, are reported in Figure 25(a–b) with $N_{YPA} = 10$ and varying N_{VA}, where N_{YPA} is the number of the YPAs and N_{VA} is the number of the VAs.

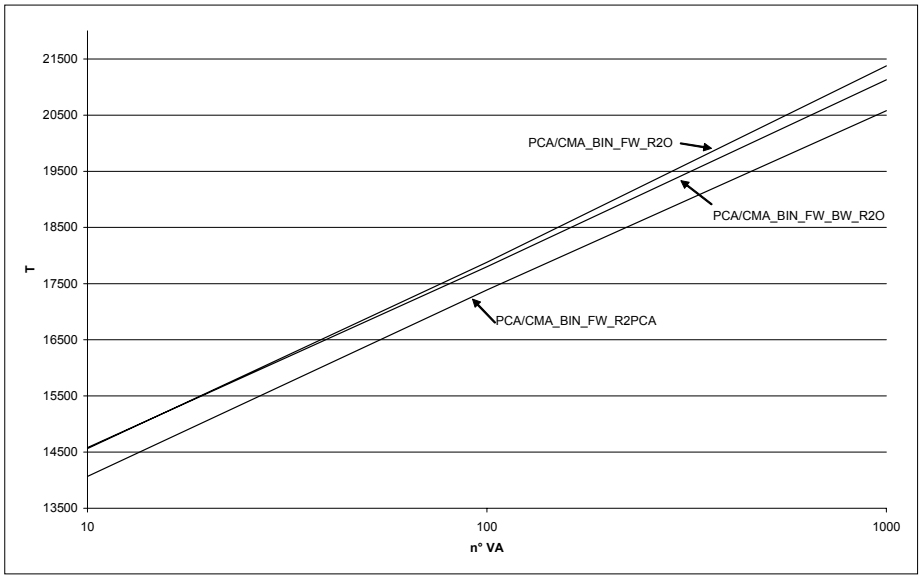
The results show that the PCA outperforms the ICA and that the PCA/CMA_FP is the better solution from the point of view of time efficiency, even though it suffers from the resource consumption issues highlighted in Section 4.3.1. It is worth saying that

the simulated PCA/CMA_FP is only an ideal implementation and that the obtained curve is a lower bound for a real implementation of the PCA/CMA_FP. Among the PCA/CMA_BINs, the PCA/CMA_BIN_FW_R2PCA exhibits better performance even though it could cause bottleneck issues at the PCA site.

Figure 25 (a) Evaluation of T_{BTC} for the five types of MCA with SP = ALL, BP = MP, N_{YPA} = 10 and variable N_{VA}; (b) Zoom in of the T_{BTC} curves of the PCA/CMA_BIN agents



(a)



(b)

5 Related work

In the following, we briefly describe some interesting approaches for the development of agent-based systems which explicitly incorporate simulation (see Table 5). In Sierra *et al.* (2004) an integrated development environment for the engineering of MASs as Electronic Institutions (EIs) is presented. This includes SIMDEI, a simulation tool which allows for the animation and analysis of the specification of the rules and protocols in an EI. In Rohl and Uhrmacher (2004) a modelling and simulation framework (DynDEVS) for supporting the development process of MASs from specification to implementation is proposed. The exploited simulation framework is JAMES, a Java-Based Agent Modelling Environment for Discrete Event Systems Specification (DEVS)-based Simulation, which aims at exploring the integration of the agents paradigm within a general modelling and simulation formalism for discrete-event systems. In Martelli *et al.* (1999) a logic-based prototyping environment for MASs, Complex Application Specification Environment Based on Logic Programming (CaseLP), is presented. CaseLP integrates simulation tools for visualising the prototype execution and for collecting the related statistics. In Fortino *et al.* (2005a; 2005b) an integrated approach, centred on GAIA and MASSIMO, for the development and validation through simulation of MASs is proposed and exemplified. In Gardelli *et al.* (2005) the authors make a preliminary study on methodological aspects of the engineering of self-organising MASs. They promote the use of formal tools, such as stochastic π-calculus process algebra, for simulating the dynamics of MASs at the early stages of design. The tool is exploited to evaluate different scenarios of an intrusion detection infrastructure for MASs based on TucSon, which detects malicious agents in an open context. Sarjoughian *et al.* (2001) presents a layered architectural framework to support agent-based system development in a collaborative, multidisciplinary engineering setting. The framework supports incremental specification, design, implementation and simulation of agent-based systems. The simulation phase is enabled by Joint Measure (JM), which is built upon DEVS/HLA, a generic HLA-compliant distributed simulation environment. Although JM affords a baseline to consider the requirements for agent development simulation environments, it is not intended to focus on agents per se. In Pavon *et al.* (2006) a simulation phase based on the agent-based simulation toolkit Repast is defined and introduced in the INGENIAS agent-oriented methodology for the development of MASs. The main objective is to support modelling and simulation of social systems.

Although the overviewed methodologies offer different approaches to the modelling and simulation of MAS, all of them use simulation to validate and evaluate the design of the MAS being developed. Moreover, few of them represent a full-fledged methodology (*i.e.*, covering all the MAS development life cycle) for the simulation-driven development of general-purpose MASs as PASSIM. Finally only PASSIM, Ingenias and EI offer visual modelling tools for supporting the development process.

Table 5 Comparison of simulation-based methodologies for MAS development

	Features		
Methodology	*Simulation purpose*	*Full-fledged methodology*	*Modelling tools*
EI	Design support	Yes	Yes
DynDEVS	Design support	No	No
GAIA/Massimo	Design support	Yes	No
JM-DEVS/HLA	Design support	No	No
Ingenias/Repast	Design support	Yes	Yes
CaseLP	Design support	No	No
Tucson/pi	Design support	No	No
PASSIM	Design support	Yes	Yes

6 Conclusion

This paper has proposed PASSIM, an agent-oriented process for the simulation-driven development of MASs. PASSIM was created through an experiment in situational method engineering according to a process-driven approach which allowed to integrate method fragments deriving from PASSI and the DSC-based simulation methodology.

The proposed case study, which is concerned with an AeM, has highlighted the efficacy of PASSIM both for the analysis and design of complex MASs and, specifically, for their simulation-oriented prototyping, which allows for the validation of functional requirements (agent behaviours and protocols) on the basis of event traces and of alternative design solutions on the basis of *ad hoc* defined performance indices.

PASSIM represents a novel contribution to the AOSE research area as it represents a new tool which promotes experimenting with the prototyping of complex MASs through simulation, to support the development of higher-quality agent-based software systems.

Currently, research efforts are underway to:

- apply PASSIM in larger case studies in order to prove the efficacy and scalability of the methodology
- enhance PASSIM for prototyping self-organising MASs
- support PASSIM in the Metameth tool, which is a method engineering tool for designing and deploying agent-oriented software development methodologies supporting collaborative and distributed design processes (Cossentino *et al.*, 2006).

References

Bernon, C., Cossentino, M. and Pavón, J. (2005) 'Agent Oriented Software Engineering', *Knowledge Engineering Review*, Vol. 20, No. 2, pp.99–116.

Brinkkemper, S., Lyytinen, K. and Welke, R. (1996) 'Method engineering: principles of method construction and tool support', *International Federation for Information Processing.*

Brinkkemper, S., Saeki, M. and Harmsen, F. (1999) 'Meta-modelling based assembly techniques for situational method engineering', *Information Systems*, Vol. 24, pp. 209–228.

Chella, A., Cossentino, M. and Sabatucci, L. (2003) 'Designing Jade systems with the support of case tools and patterns', *Exp Journal*, Vol. 3, No. 3, pp.86–95.

Christie, A.M. (1999) 'Simulation – an enabling technology in software engineering', *Technical Article, The Software Engineering Institute (SEI)*, Carnegie Mellon University, http://www.sei .cmu.edu/publications/articles/christie-apr1999/christie-apr1999.html.

Cossentino, M. (2005) 'From requirements to code with the PASSI methodology', in B. Henderson-Sellers and P. Giorgini (Eds.) *Agent-Oriented Methodologies*, Hershey, PA: Idea Group Inc.

Cossentino, M., Garro, A., Gaglio, S. and Seidita, V. (2007) 'Method fragments for agent design methodologies: from standardization to research', *Int. J. Agent-Oriented Software Engineering*, to appear.

Cossentino, M., Sabatucci, L. and Chella, A. (2003) 'A possible approach to the development of robotic multi-agent systems', *Proc. of IEEE/WIC Conf. on Intelligent Agent Technology (IAT'03)*, Halifax, Canada.

Cossentino, M., Sabatucci, L., Seidita, V. and Gaglio, S. (2006) 'An agent oriented tool for new design processes', *Proc. of Fourth European Workshop on Multi-Agent Systems (EUMAS'06)*, Lisbon, Portugal.

Feldman, S. (2000) 'Electronic marketplaces', *IEEE Computing*, Vol. 4, July–August, pp.93–95.

Fortino, G., Garro A., Mascillaro, S. and Russo, W. (2007) 'ELDATool: a statecharts-based tool for prototyping multi-agent systems', *Proc. of the Workshop on Objects and Agents (WOA'07)*, Genova, Italy, 24–25 September.

Fortino, G., Garro, A. and Russo, W. (2005a) 'A discrete-event simulation framework for the validation of agent-based and multi-agent systems', *Proc. of the Workshop on Objects and Agents (WOA'05)*, Camerino, Italy, 14–16 November.

Fortino, G., Garro, A. and Russo, W. (2005b) 'An integrated approach for the development and validation of multi agent systems', *Computer Systems Science & Engineering*, Leicester, UK: CRL Publishing Ltd., July, Vol. 20, No. 4, pp.94–107.

Fortino, G., Russo, W. and Zimeo, E. (2004) 'A statecharts-based software development process for mobile agents', *Information and Software Technology*, Amsterdam, the Netherlands: Elsevier, Vol. 46, No. 13, pp.907–921.

Foundation for Intelligent Physical Agents (FIPA) (2001) *FIPA RDF Content Language Specification. Foundation for Intelligent Physical Agents*, Document FIPA XC00011B (2001/08/10), http://www.fipa.org/specs/ fipa00011/XC00011B.html.

Foundation for Intelligent Physical Agents (FIPA) (2002) *FIPA Agent Management Support for Mobility Specification*, Document FIPA DC00087C (2002/05/10).

Gardelli, L., Viroli, M. and Omicidi, A. (2005) 'On the role of simulation in the engineering of self-organising systems: detecting abnormal behaviour in MAS', *Proc. of Workshop on Objects and Agents (WOA'05)*, Camerino, Italy, pp.85–90.

Guttman, R.H., Moukas, A.G. and Maes, P. (1998) 'Agent-mediated electronic commerce: a survey', *The Knowledge Engineering Review*, Vol. 13, pp.147–159.

Harel, D. and Gery, E. (1997) 'Executable object modelling with statecharts', *IEEE Computer*, Vol. 30, No. 7, pp.31–42.

Harmsen, F. and Brinkkemper, S. (1995) 'Design and implementation of a method base management system for a situational CASE environment', *Proceedings of the 2nd Asia-Pacific Software Engineering Conference (APSEC'95)*, Brisbane: IEEE CS Press, pp.430–438.

Henderson-Sellers, B. (2003) 'Method engineering for OO systems development', *Communications of the ACM*, Vol. 46, No. 10, pp.73–78.

Leung, C.S., Sum, J., Shen, H., Wu, J. and Young, G. (2004) 'Analysis and design of an agent searching algorithm for e-marketplaces', *Cluster Computing*, Vol. 7, pp.85–90.

Luck, M., McBurney, P. and Preist, C. (2004) 'A manifesto for agent technology: towards next generation computing', *Autonomous Agents and Multi-Agent Systems*, Vol. 9, No. 3, pp.203–252.

Maes, P., Guttman, R. and Moukas, A. (1999) 'Agents that buy and sell: transforming commerce as we know it', *Communications of ACM*, Vol. 42, No. 3, pp.81–91.

Martelli, M., Mascardi, V. and Zini, F. (1999) 'Specification and simulation of multi-agent systems in CaseLP', in M.C. Meo and M. Vilares-Ferro (Eds.) *Proc. of Appia-Gulp-Prode Joint Conf. on Declarative Programming*, L'Aquila, Italy, pp.13–28.

Mayrhauser, A. (1993) 'The role of simulation in software engineering experimentation', *Experimental Software Engineering Issues: Critical Assessment and Future Directions, Proceedings of the International Workshop*, LNCS 706, Dagstuhl Castle, Germany, 14–18 September 1992.

Pavon, J., Sansores, C. and Gomez-Sanz, J. (2006) 'Modeling of social systems with Ingenias', *Proc. of 1st Workshop on Multi-Agent Systems and Simulation (MAS&S'06)*, Palermo, Italy.

Ralyté, J. and Rolland, C. (2001) 'An assembly process model for method engineering', *Proceedings of the 13th Conference on Advanced Information Systems Engineering, CAISE01*, Interlaken, Switzerland, pp.267–283.

Resource Description Framework (RDF) (1999) *Model and Syntax Specification*, W3C Recommendation, 22 February, http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/.

Ripper, P.S., Fontoura, M.F., Maia Neto, A. and de Lucena, C.J.P. (2000) 'V-Market: a framework for agent e-commerce systems', *World Wide Web*, Vol. 3, pp.43–52.

Rohl, M. and Uhrmacher, A.M. (2004) 'Controlled experimentation with agents – models and implementations', *Proc. of 5th Int'l Workshop Engineering Societies in the Agents World*, Toulouse, France, 20–22 October.

Royce, W. (1970) 'Managing the development of large software systems', *Proceedings of IEEE WESCON*, 26 August, pp.1–9.

Sarjoughian, H.S., Zeigler, B.P. and Hall, S.B. (2001) 'A layered modeling and simulation architecture for agent-based system development', *Proceedings of the IEEE*, Vol. 89, No. 2, pp.201–213.

Sierra, C., Rodríguez-Aguilar, J.A., Noriega, P., Esteva, M. and Arcos, J.L. (2004) 'Engineering multi-agent systems as electronic institutions', *Novática*, Vol. 170.

Software Process Engineering Metamodel Specification (SPEM) (2006) Version 2.0, 06-11-03, Object Management Group Inc.

Uhrmacher, A.M. (2002) 'Simulation for agent-oriented software engineering', *Proc. of 1st Int'l. Conference on Grand Challenges for Modeling and Simulation*, San Antonio, Texas, USA, 27–31 January.

Wang, Y., Tan, K-L. and Ren, J. (2002) 'A study of building internet marketplaces on the basis of mobile agents for parallel processing', *World Wide Web: Internet and Web Information Systems*, Vol. 5, No. 1, pp.41–66.

Wooldridge, M. (2002) *An Introduction to Multiagent Systems*, John Wiley & Sons Ltd.

Zambonelli, F. and Omicini, A. (2004) 'Challenges and research directions in agent oriented software engineering', *Autonomous Agents and Multi-Agent Systems*, Vol. 9, No. 3, pp.253–284.