SIMULATION-DRIVEN DEVELOPMENT OF MULTI-AGENT SYSTEMS

Giancarlo Fortino, Alfredo Garro, Wilma Russo
Dipartimento di Elettronica, Informatica e Sistemistica
Università della Calabria
Via P.Bucci, cubo 41C, 87036 Rende (CS)
Italy
E-mail: {g.fortino, garro, w.russo}@unical.it

Roberto Caico, Massimo Cossentino, Francesco Termine
ICAR
Italian National Research Council
Viale delle Scienze, 90128 Palermo
Italy
E-mail: {caico, cossentino, termine}@pa.icar.cnr.it

KEYWORDS
Multi-Agent Systems, Discrete-Event Simulation, Agent Oriented Software Engineering

ABSTRACT

This paper presents a simulation-driven development process for multi-agent systems (MAS) which integrates a Statecharts-based simulation methodology into the well known and established PASSI methodology. It can be effectively used as an experimental tool in the context of the Agent Oriented Software Engineering for quantifying the benefits of using simulation for MAS development. To exemplify this process and demonstrate its effectiveness, a case study concerning with the design and simulation of a complex MAS is defined and detailed.

1. INRODUCTION

Simulation is being greatly applied in many industrial fields, such as aerospace, automotive or energy production, but its application in support of software products and processes is to date still under estimated. Despite of its limited exploitation in software engineering, Simulation has been recognized to be an effective tool to support software engineering experimentations involving requirements management, project management, training, process improvement, architecture and COTS (Commercial Off-The-Shelf) 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) which aims at identifying and defining models and techniques suitable for the development of complex software systems in terms of MASs (Multi-Agent Systems), we wonder if Simulation could play a more strategic role in the development of MASs than that 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 complexity of traditional software systems. A MAS is a system composed of several agents, capable of reaching goals that are difficult to achieve by an individual system (Woolridge 2002). MASs can manifest self-organization and complex behaviors 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 under-development 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 lifecycle of MASs: requirements capture, analysis, design, implementation, deployment, and testing. To date a few MAS development processes have been proposed in the literature (Electronic Institutions (Sierra et al. 2004), DynDEVS/James (Rohl and Uhrmacher 2004), CaseLP (Martelli et al. 1999), GAIA/MASSIMO (Fortino et al. 2005a), TuCSon/Simulation (Gardelli et al. 2005), Joint Measure (Sarjoughian et al. 2001), etc) which incorporate Simulation to support the design phase of the MAS development lifecycle 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 need to be carried out in the aforementioned direction and in further directions encompassing all the phases of the MAS development lifecycle. The major benefits would be product quality improvement and project risk minimization. These would derive from the use of Simulation in pinning down MAS requirements early in the development lifecycle, in testing out alternate modifications of requirements, in safely examining alternate architectures and designs, and in gaining insights with timing, resource usage and bottlenecking.
In this paper we propose a simulation-driven development process which is obtained by integrating a Statecharts-based simulation methodology for MASs (Fortino et al. 2005a, Fortino et al. 2005b) with PASSI, a well-known development process for MASs (Cossentino 2005). This allows us, on one hand, to enrich PASSI with the potential benefits deriving from the exploitation of Simulation and, from another hand, to concretely experiment with a process supporting simulation-driven development of MAS. The obtained development process is exemplified through a case study concerning with the design and simulation of a MAS which represents a consumer-driven e-marketplace (CEM). In particular, the simulation phase allows for the validation of the correct behavior of the CEM under-development and for the evaluation of the CEM efficiency, in terms of completion time for buying a product, and efficacy, in terms of probability of buying a product at the desired price.
The rest of the paper is organized as follows. Section 2 describes the proposed agent-oriented simulation-driven development process. Section 3 is devoted to detail the proposed case study. Section 4 discusses some related agent-based design and development approaches incorporating simulation. Finally some conclusions are drawn and directions of future research briefly elucidated.

2. AN AGENT-ORIENTED SIMULATION-DRIVEN DEVELOPMENT PROCESS

In this section we present an agent-oriented simulation-driven development process for building MAS. It is obtained through the integration of MASSIMO (Multi-Agent System SIMulator framewOrk) and its supporting Statecharts-based simulation methodology (Fortino et al. 2005a) into PASSI (Process for Agent Societies Specification and Implementation), a development process for MAS (Cossentino 2005). The obtained development process can use simulation to support the following phases of PASSI: *system requirements*, *agent society* and *agent implementation*. In particular, in this paper, we concentrate on the simulation of the Agent Implementation Model, i.e. the work product of the agent implementation phase describing the complex structure and behaviour of the MAS under-development, both for validation and for performance evaluation purposes. In the following subsections we first provide a brief description of PASSI and the Statecharts-based simulation methodology, and, then, we present their integration.

2.1. PASSI (Process for Agent Societies Specification and Implementation)

The PASSI methodology is a step-by-step requirements-to-code methodology for designing and developing multi-agent societies. It adopts design models and concepts from the UML that is adapted in order to represent the different elements and abstractions of a multi-agent system. The methodology is supported by the PASSI Toolkit (PTK), a Rational Rose plug-in, and by a repository of agent patterns.
In PASSI, during the initial steps of the design, an agent is seen as an autonomous entity capable of pursuing an objective through its autonomous decisions, actions and social relationships. This helps in preparing a solution that is later implemented referring to the agent as a significant software unit. An agent may undertake several functional roles during interactions with other agents to achieve its goals. A role is a collection of tasks performed by the agent in pursuing a sub-goal or offering some service to the other members of the society. A task is defined as a purposeful unit of individual or interactive behavior. Each agent has a representation of the world in terms of an ontology that is also referred to in all the messages the agents exchange.
PASSI is composed of the following five models regarding the different abstraction levels of the process (see Fig. 1):

- *System Requirements Model*. The initial part of this model is similar to other common object-oriented methodologies (requirements analysis phase), then an agent-based solution to the problem is drafted by assigning system functionalities to agents.
- *Agent Society Model*. This describes the details of the system solution in terms of agent society concepts like ontology, communications and roles.
- *Agent Implementation Model*. The previous models are used to obtain a detailed description of the agent society in terms of both structure and behavior that can be used to produce the code of the system.
- *Code Model*. In order to streamline and speed up the development of a new system, code is partially obtained from the application of patterns. A conventional code completion activity is then carried out.
- *Deployment Model*. Mobile agents require that a specific attention is paid to the specification of their needs in terms of both software environments (e.g., libraries available in the host platform), hardware capabilities and performance (e.g., amount of available network bandwidth); these are the issues defined in the deployment model.

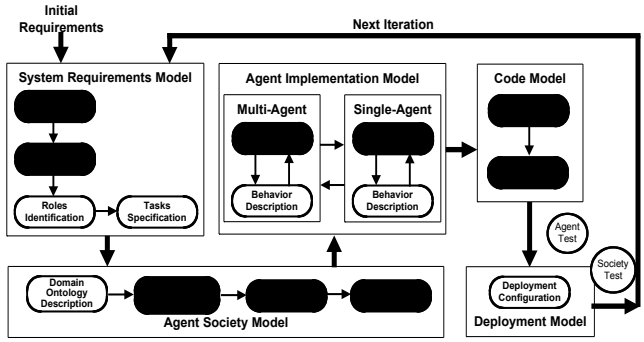


Figure 1: The different steps and models of PASSI

2.2. A Statecharts-based simulation methodology for multi-agent systems

The simulation methodology (Fortino et al. 2005a) is based on the following three iterable phases: *Modeling*, *Coding* and *Simulation* of the MAS under-development (see Figure 2).

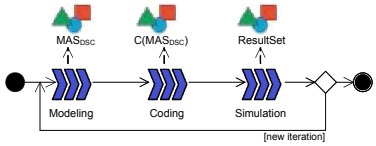


Figure 2: The simulation process

The *Modeling* phase is enabled by the Distilled StateCharts (DSCs) formalism (Fortino et al. 2004) which supports the specification of the behavior 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 behavior 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 behavior 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 lifecycle (FIPA 2001) (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).
The *Coding* (or prototying) phase 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 $E_{Beh}(AT_i)$.

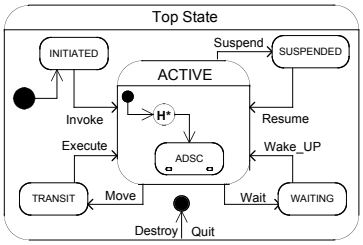


Figure 3: The FIPA-compliant DSC template

The *Simulation* phase is supported by MASSIMO (Multi-Agent System SIMulator framewOrk), a Java-based discrete-event simulation framework for MAS which allows for the validation and evaluation of:

- the dynamic behavior (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 behaviors of applications and systems based on agents.

In particular the architecture of MASSIMO (Fortino et al 2005b) is composed of four basic layers:

(i) *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;

(ii) *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 signaling messages (MSG).

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

(iv) *User*, which makes it 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 MASSIMO, a simulator program can be implemented and executed to obtain a *ResultSet* containing validation traces and performance parameter values. The validation of agent behaviors and interactions is carried out on execution traces automatically generated, whereas the performance evaluation relies on the specific MAS to be analyzed; the performance evaluation parameters are therefore set ad-hoc. The *ResultSet* can notably be used to feed back the *Modeling* phase.

2.3. Integrating MASSIMO into PASSI

Although the simulation methodology overviewed in the previous section could be used to validate the work products of the system requirements, agent society and agent implementation phases of PASSI, it is currently used for the validation of the Agent Implementation Model (AIM). PASSI is therefore enhanced with a further "step" involving the simulation of the AIM which must previously be translated into a MAS_{DSC}.

The semi-automatic translation process of the AIM (see Fig. 1) into the MAS_{DSC} is carried out as follows:

- The ATs are derived from the agent types of the Multi-Agent Structure Definition (MASD) through a one-to-one mapping.
- The interactions in terms of events exchanged between the ATs are derived from the Multi-Agent Behavior Definition (MABD).
- The $Beh(AT_i)$ $\forall i$ is derived from the SASD (Single-Agent Structure Definition) and the SABD (Single-Agent Behavior Definition) of the i-th agent type.

A translation example based on the proposed case study will be presented in section 3.2. After the simulation phase, the designers can either proceed with the remaining part of the PASSI process, if they want to implement the software final release, or use the results of the simulation to feedback the *System Requirement* phase and/or the *Agent Society* phase.

3. A CASE STUDY

This section shows the application of the proposed approach to the analysis and design of a Consumer-driven E-Marketplace (CEM) system. A CEM system is a distributed software system which provides e-commerce services to end-users (or consumers) which drive the exchange of goods within the e-Marketplace. In particular, users according to their needs, browse the e-Marketplace, search for the vendors offering a given product, evaluate the vendors' offers, contract product price with the vendors and, finally, decide to buy a product from a selected vendor. The payment phase is supported by an e-cash-based system mediated by a bank.

In subsection 3.1 PASSI is used to design the CEM. Subsection 3.2 shows the translation of a single agent behaviour description into a DSC model. Finally subsection 3.3 shows the simulation phase for a given CEM scenario.

3.1 Designing the CEM system with PASSI

In the following subsections the obtained system requirements, agent society and agent implementation models will be described.

3.1.1 The System Requirements Model

The *System Requirements Model* is a model of the system requirements in terms of agency and purpose. The methodology is use case driven and starts with the requirements analysis, where the designer models the system as a set of use case diagrams. Some of these diagrams, the Domain (Requirements) Description diagrams, are drawn to represent the actors and the use cases identified for the system. A use case represents a portion of the system behavior while an actor is an external entity interacting with the system; we identified the actors User, Vendor and Bank.

In PASSI, each agent receives the responsibility for a part of the functionalities of the whole system; this is represented in a use case diagram, called Agent Identification (AId) diagram, by grouping some of the use cases within a package and giving it the name of the agent.

Figure 4 depicts the AId diagram for our system which includes the following identified agents:

- *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 buying the product according to a specific buying policy.
- *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.
- *Vendor Agent (VA)* represents the vendor of specific goods.
- *Mobile Consumer Agent (MCA)* is an autonomous mobile agent dealing with searching, contracting, evaluation, and payment of goods.
- *Access Point Agent (APA)* represents the entry point for the e-marketplace. It accepts requests for buying a product from a registered UAA and fulfils them by generating a specific MCA.
- *Bank Agent (BA)* represents a reference bank of MCA and VA.

In the reported diagram, these agents are displayed as packages containing the use cases coming from the Domain (Requirements) Description Diagram that has been omitted because of space concerns. Each agent is responsible for accomplishing the functionalities associated with the use cases included in its package. Because of the specific nature of this diagram (a functional view), we cannot describe here agents interactions; this consideration finds an exception in the *communicate* relationship substituting the *include/extend* relationships occurring between use cases of different agents (being an agent an autonomous entity, it makes no sense to design an *include* dependency between two different ones).

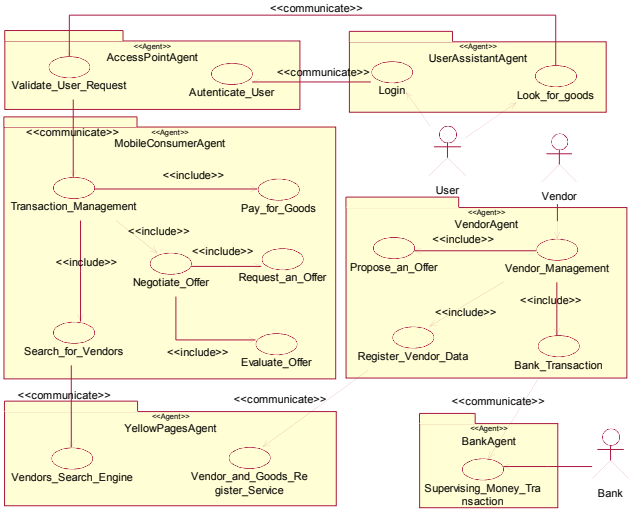


Figure 4: The AId diagram for the proposed case study

Once all the use cases have been assigned to agents that will be responsible for accomplishing them, the designer can explore the scenarios in which these agents will be involved. We usually do it with a set of UML sequence diagrams (Role Identification diagrams); in these diagrams each agent may be involved in several different activities and may appear more than once in each scenario playing different roles.

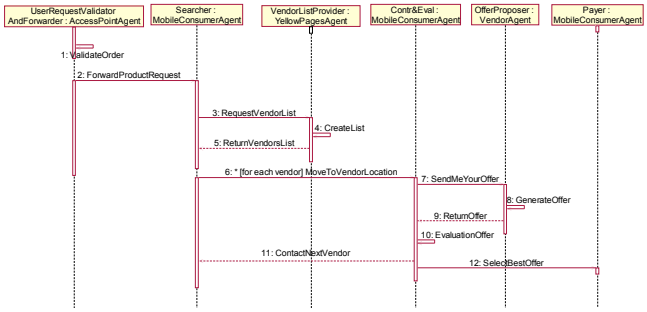


Figure 5: A portion of a RId diagram regarding a specific-product vendors search scenario

An example of a Role Identification (RId) diagram is shown in Figure 5 where the APA, playing the role of *UserRequestValidatorAndForwarder*, after validating the order, forwards it to the MCA, playing the *Searcher* role; hence the MCA asks for the vendor list to the YPA. Once the MCA gets the list, it contacts all the VAs and asks them for their offers.

The initial description of the dynamic behavior of each agent is the last step of the System Requirements Model. This phase is performed with a set of Task Specification Diagrams (one for each identified agent). The Task Specification Diagram is a UML activity diagram that represents the agent activity plan using two swim-lanes (see Figure 6): the right-hand contains a collection of roles including activities, while the left-hand reports some roles from other agents involved in interactions with this one; in this diagram the activities performed by the agent within each of its roles (*Searcher, Contr&Eval, Payer, Reporter*) are hidden.

The example reported in Figure 6, regards the MCA that is involved in searching the vendors list through a query to the YPA (*Searcher* role), then in the contracting and evaluation phase (*Contr&Eval* role) with the VA, in buying the product from the best bidder (*Payer* role) and in reporting (*Reporter* role) the transaction results to the UAA. At this point the MCA can play again its first role or it can be terminated.

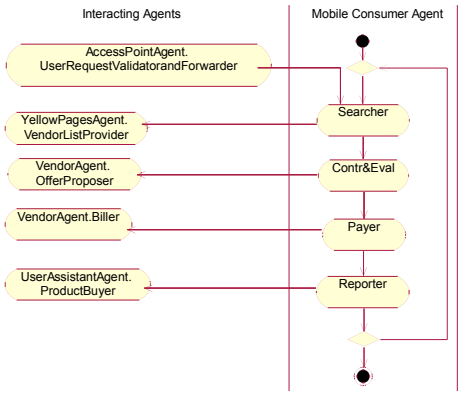


Figure 6: The Task Specification diagram for the MCA

3.1.2 The Agent Society Model

The next PASSI model is the *Agent Society Model* that represents social interactions and dependencies among agents involved in the solution. It begins with the ontology design that is performed in the Domain Ontology Description (DOD) phase with the use of a class diagram. A DOD diagram 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) encoding of the ontological structure.

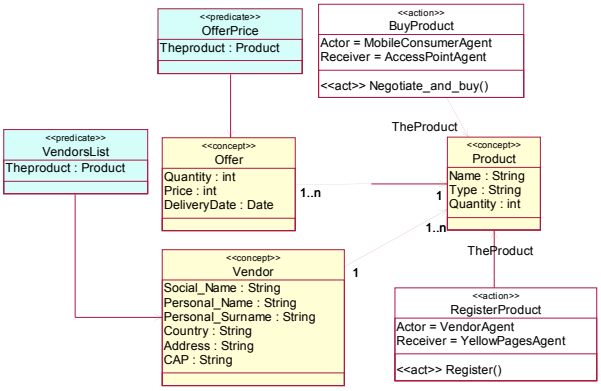


Figure 7: A portion of the DOD diagram

Figure 7 shows a portion of the DOD diagram obtained for the case study, where we can see some of the concepts, predicates and actions used to define the problem domain. 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 and its outcome received by the YPA.

The Communication Ontology Description (COD) is a class diagram that shows all agents and all of their communications (relationships among agents). This diagram is drawn starting from the results of the AId phase. A class is introduced for each agent, and an association is introduced for each communication between two agents. Obviously, according to the principles of an iterative/incremental design process, in further refinement communications can be added, merged or removed as a consequence of the arising needs. Being communications a way to exchange knowledge, it is also important to introduce the proper data structure (coming from the entities described in the DOD) in each agent in order to store it. 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 RId phase. Each communication is characterized by three attributes, (Ontology, Agent Interaction Protocol and Content Language) which we group into an association class. This is the characterization of the communication itself and its name is used to uniquely identify it (this communication can have, obviously, several instances at runtime).

In Figure 8 an example of COD diagram is reported. It represents three agents (APA, VA, MCA) and two communications among them (Forward_Product_Request, Offer_Request). In particular, the Offer_Request communication happens when, in the scenario reported in Figure 5, the MCA asks the VA for the best offer.

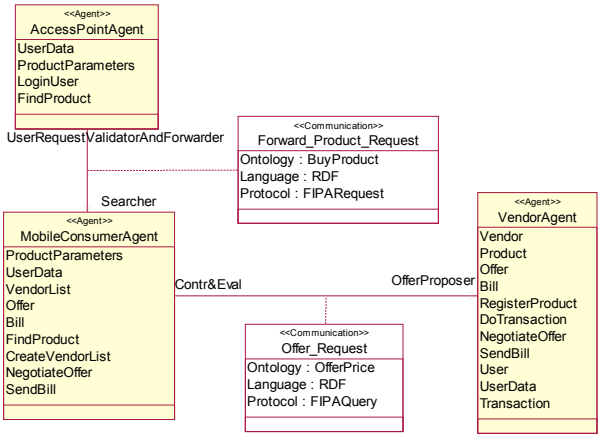


Figure 8: A portion of the COD diagram

This communication refers to the OfferPrice predicate from the ontology of Figure 7 and adopts the FIPAQuery agent interaction protocol and the RDF content language.

Roles played by agents during the interaction (as described in the RId diagrams) are reported at the beginning and the end of the association line.

As it has already been discussed in previous sub-section, PASSI roles are initially identified in the AId diagrams. Their definition is then completed with the Role Description (RD) diagram that is a UML class diagram in which classes are used to represent roles; each role uses several elementary tasks to implement its complex behavior; finally, roles are grouped in packages representing agents.

The Agent Society Model ends with the Protocol Description phase which is required only when the FIPA standard protocols are not sufficient to solve some communication problem (this is not the case for our case study).

3.1.2 The Agent Implementation Model

The *Agent Implementation Model* is a model of the solution architecture. It is composed of two different phases, each performed at both the multi- and single-agent level of abstraction. The multi-agent level deals with the agent society and reports low details about agent implementation; however, it fittingly documents the overall structure of the system (behaviors of each agent, communications, etc.).

The single-agent level of abstraction focuses on the implementation details of each agent and specifies whatever is needed in order to prepare the coding phase.

The two phases that are performed at the multi- and single-agent levels are:

- Agent Structure Definition (ASD); that uses conventional class diagrams to describe the structure of solution agent classes;
- Agent Behavior Description (ABD); that uses activity diagrams or state-charts to describe the behavior of individual agents.

In the *Multi-Agent Structure Definition* (MASD) diagram, automatically generated by the PTK tool on the basis of the previous diagrams, the focus is on the general architecture of the system. The MASD diagram is an overview of the multi-agent system from the structural point of view. In this diagram, agents are represented as classes with their behaviors in the operation compartment and attributes specifying the agent knowledge.

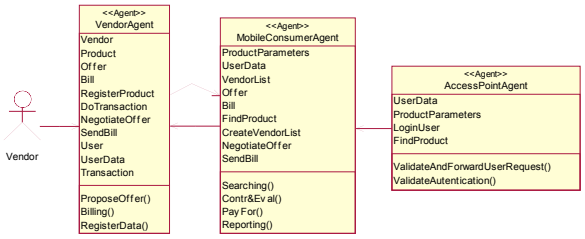


Figure 9: A portion of the MASD diagram

In Figure 9 we report the portion of the MASD describing the structure of the VA, MCA and APA agents. It is worth to note that the VA is in relationship with an (human) actor; this is an extension of UML that we consider useful to represent in a unique diagram all the agents relationships (communications and GUI-based interactions with the user).

The agent behavior at the multi-agent level is described by the Multi-Agent Behavior Description (MABD) diagram. This is a UML activity diagram used to illustrate the dynamics of the system during the agents' lifecycle. In the diagram, the involved agents and their tasks are represented with swim-lanes, while operations are displayed as activities. In this diagram, transitions among activities represent events like method invocations (if relating activities in the same swim-lane), new behavior instantiations/invocations (if relating activities of the same agent but in different swim-lanes) or messages (if activities from two different agents are involved). Figure 10 reports a portion of the obtained MABD diagram which illustrates the activities occurring during the Request communication between MCA and YPA and the Query communication between MCA and VA. In particular, it describes the request of the vendors list from the MCA to the YPA; then, the MCA moves to the VA location and begins the contract phase by asking for an offer which is soon after evaluated.

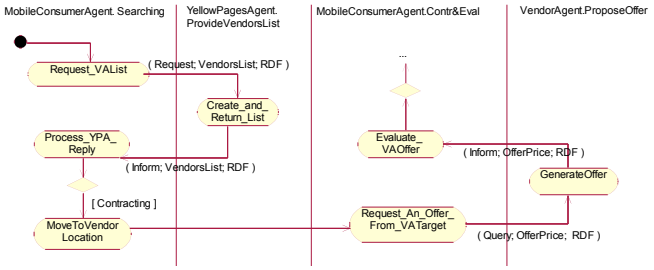


Figure 10: A portion of the MABD diagram with some interactions among MCA, YPA and VA

Although this representation is very useful and gives a complete overview of the MAS, it is not sufficient to detail the algorithm implemented in each of the agents within the activities. This further refinement step is usually done at the single-agent level through the Single-Agent Behavior Description, which is an activity diagram in which the swimlanes represent the tasks performed by the agent during its lifecycle. Figure 11 shows the general SABD of the MCA, which should be self-explanatory.

This is the point where we should consider whether to implement the system or to simulate it. For simulation purposes, we have to translate the SABD of each agent into a DSC. In the next subsection we therefore use the DSC formalism to detail the SABD of a specific MCA.

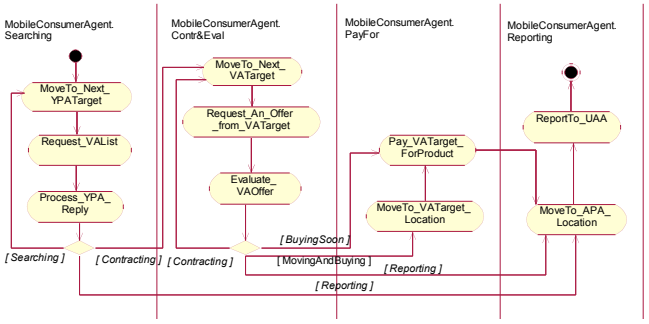
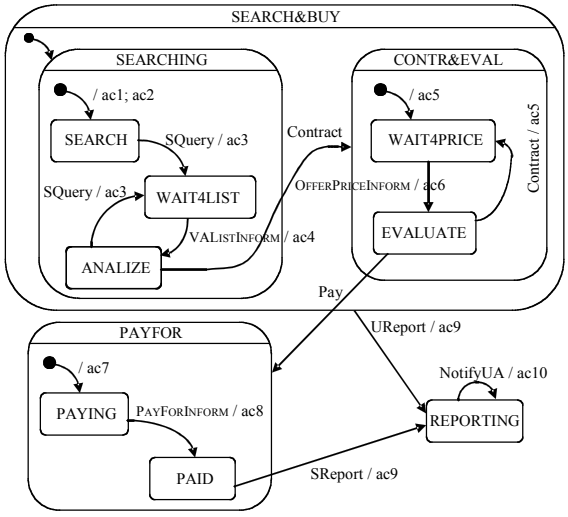


Figure 11: The SABD diagram for the MCA

3.2. The DSC-based SABD of an MCA

On the basis of the SABD shown in Figure 11, two types of DSC-based MCA have been implemented: (i) Itinerary Consumer Agent (ICA), which performs the *Searching* and *Contr&Eval* phases by sequentially moving from one location to another within the e-Marketplace; (ii) Parallel Consumer Agent (PCA), which performs the *Searching* and *Contr&Eval* phases by means of a set of parallel mobile agents called workers (Fortino et al. 2005a).

Figure 12 shows the ADSC (see section 2.2) of the ICA obtained from the SABD of the MCA (see Fig. 11) and from the MABD (see Fig. 10).




```
ac1: nextYPA=0;
ac2: YPATarget=(MAOId)ypaList.elementAt(nextYPA++);
     generate(new Move(self(), YPATarget.getCurrLocation()));
     generate(new SQuery(self()));
ac3: generate(new VALISTREQUEST(self(), YPATarget, vaListQuery));
ac4: VALISTINFORM reply = (VALISTINFORM)mevent;
     Proc proc = processYPAReply(reply);
     if (proc.continueSearching()) ac2();
     else if (proc.noVendors()) sa2();
     else { nextVA=0; sa1(); }
sa1: VATarget=(MAOId)vaList.elementAt(nextVA++);
     generate(new Move(self(), VATarget.getCurrLocation()));
     generate(new Contract(self()));
sa2: generate(new UReport(self()));
ac5: generate(new OFFER PRICE QUERY (self(), VATarget, priceQuery));
ac6: OFFER PRICE INFORM offer = (OFFER PRICE INFORM)mevent;
     Eval eval = evaluateVAOffer(offer);
     if (eval.buySoon()) generate(new Pay(self()));
     else if (eval.moveAndBuy()){
       generate(new MAOMove(self(), VATarget.getCurrLocation()));
       generate(new Pay(self()));
     else if (eval.noBuy()) sa2();
     else sa1();
ac7: bills = prepareBills(price); generate(new PAYFORREQUEST(self(), VATarget, bills)
ac8: nbills = nbills - eval.price(); generate(new SReport(self()));
ac9: generate(new Move(self(), self().getHomeLocation()));
     generate(new NotifyUA(self()));
ac10 reportTransactionResult();
```

Figure 12: The ADSC of the ICA

In the ADSC of the ICA the events, which can be internal (i.e. self-driving the agent behavior) or external (i.e.

targeting another agent), are generated through the primitive `generate(<mevent>(<param>))`, where `mevent` is an event instance and `param` is the list of formal parameters of `mevent`. In addition, events are asynchronously received and processed according to a run-to-completion semantics (i.e. an event can be processed only if the processing of the previous event has been fully completed) (Fortino et al. 2004).

The names of the composite states of the ADSC corresponds to the names of the tasks of the MCA shown in the related SABD. For the sake of modularity the SEARCHING and CONTR&EVAL states are embodied into the SEARCH&BUY state.

The activities reported in the SABD are implemented by the action chains of the ADSC; the association between activities and action chains is reported in Table 1.

Table 1: Association between the SABD activities of the MCA and the ADSC action chains of the ICA

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	ac6
Pay_VATarget_ForProduct	ac7, ac8
MoveTo_APA_Location	ac9
ReportTo_UAA	ac10

The messages that the MCA exchanges with the YPA, VA, and UAA agents during its lifecycle, reported in the MABD, are implemented through external events in the ADSC; the association between messages and events is reported in Table 2 for the interactions with YPA and VA.

Table 2: Association between the MABD messages of the MCA and the ADSC events of the ICA

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

3.3. Simulation Phase

The simulation phase, supported by MASSIMO, allows for the validation of the system's requirements, the behavior of each agent type and the related agent interactions, and the evaluation of two specific performance indexes:

- the *Buy Task Completion Time* (T_{BTC}), which is defined as $T_{BTC}=T_{CREATION}-T_{REPORT}$ where, $T_{CREATION}$ is the creation time of the MCA and T_{REPORT} is the reception time of the MCA report;
- the *Probability of Successful Buy* (P_{SB}), which is defined as the probability of successfully buying a desired product within the e-Marketplace.

In particular, the simulation scenario 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 modeling the communication delay (δ) as a lognormally distributed random variable.

Moreover, to compare the simulation results obtained for the evaluated performance indexes with the results of well defined analytical models, the simulated e-marketplace was a quite simple e-marketplace in which we supposed that 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}), and 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}.

Given the above described scenario, 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 3), moreover it is supposed that $P_{MAX}=PP_{MAX}$ so always guaranteeing a successful purchase at the best price.

The results, obtained adopting a YPA organization in which the YPAs are logically connected as a binary tree, are reported in Figure 13 with $N_{YPA}=\{10, 100\}$ and varying N_{VA}, where N_{YPA} is the number of the YPA agents and N_{VA} is the number of the VA agents. The simulation results agree with the results obtainable applying the analytical model reported in (Wang et al. 2002) and confirm that the PCA, due to its parallel dispatching mechanism, outperforms the ICA when N_{VA} increases.

Table 3. Searching and Buying Policies of MCA.

SEARCHING POLICY (SP)	
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)	
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

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

$P_{SB}=1-[(PP_{MAX}-P_{MAX})/(PP_{MAX}-PP_{MIN}+1)]^V$, where: V is the number of VA agents 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 for the user), whereas $PP_{MAX}-PP_{MIN}+1$ 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 14. 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 $[1..N_{VA}]$.

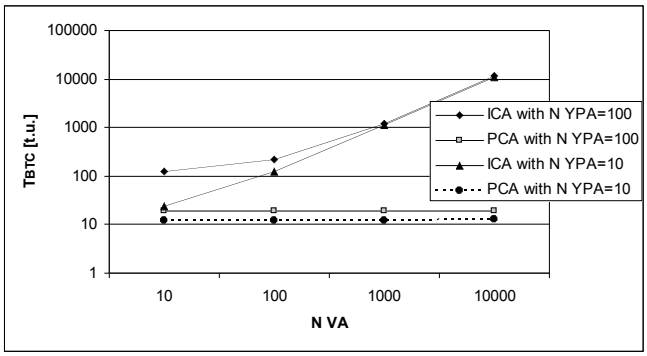


Figure 13: Evaluation of T_{BTC} for an MCA with SP=ALL, BP=MP, N_{YPA} ={10, 100} and variable N_{VA}

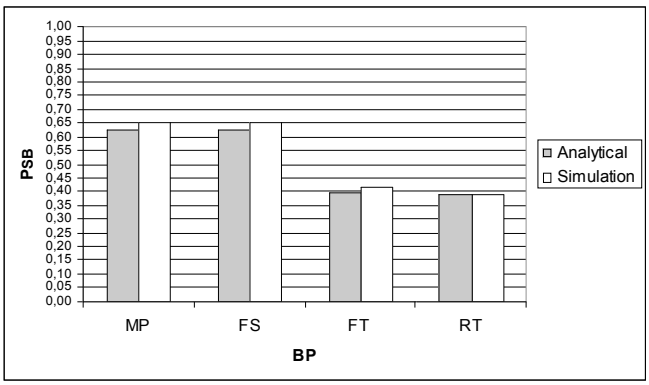


Figure 14: Evaluation of P_{SB} for the defined BPs with PP_{MAX}=200, PP_{MIN}=100, P_{MAX}=PP_{MIN}, and N_{VA}=100

4. CONCLUSION

This paper has proposed and exemplified through a case study an agent-oriented simulation-driven development process obtained by enhancing PASSI with a simulation step based on a simulation methodology centered on Distilled Statecharts and related tools. The resulting process represents a novel contribution to the AOSE research area as it is a new tool which promotes experimenting with the design and the simulation of complex MASs to support the development of higher-quality agent-based software systems. Currently our research efforts are geared at applying the simulation-driven development process for the construction and analysis of self-organizing MASs.

REFERENCES

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/oublications/articles/christie-apr1999/christie-apr1999.html

Cossentino, M. 2005. "From Requirements to Code with the PASSI Methodology". In *Agent-Oriented Methodologies*, B. Henderson-Sellers and P. Giorgini (eds). Idea Group Inc., Hershey, PA, USA.

Fortino, G.; A. Garro; and W. Russo. 2005. "An Integrated Approach for the Development and Validation of Multi Agent Systems". In *Computer Systems Science & Engineering*, 20(4), pp. 94-107, CRL Publishing Ltd., Leicester (UK), Jul.

Fortino, G.; A. Garro; and W. Russo. 2005. "A Discrete-Event Simulation Framework for the Validation of Agent-based and Multi-Agent Systems". In *Proc. of the Workshop on Objects and Agents* (WOA'05), Camerino (Italy), Nov 14-16.

Fortino, G.; W. Russo; and E. Zimeo. 2004. "A Statecharts-based Software Development Process for Mobile Agents". In *Information and Software Technology*, 46(13), pp.907-921, Elsevier, Amsterdam, The Netherlands.

FIPA (Foundation for Intelligent Physical Agents). 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

Gardelli, L.; M. Viroli; and A. Omicini. 2005. "On the Role of Simulation in the Engineering of Self-Organising Systems: Detecting Abnormal Behaviour in MAS". In *Proc. of Workshop on Objects and Agents* (WOA'05), Camerino (Italy), pp. 85-90.

Harel, D. and E. Gery. 1997. "Executable Object Modelling with Statecharts". *IEEE Computer*, 30(7), pp. 31-42.

Luck, M.; P. McBurney; and C. Preist. 2004. "A Manifesto for Agent Technology: Towards Next Generation Computing". *Autonomous Agents and Multi-Agent Systems*, 9(3), pp. 203-252.

Martelli M.; V. Mascardi; and F. Zini. 1999. "Specification and Simulation of Multi-Agent Systems in CaseLP". *Proc. of Appia-Gulp-Prode Joint Conf. on Declarative Programming*, L'Aquila, Italy. M.C. Meo and M. Vilares-Ferro (eds), pp. 13-28.

Mayrhauser, A. 1993. "The Role of Simulation in Software Engineering Experimentation". H. Dieter Rombach, Victor R. Basili, Richard W. Selby (eds.): *Experimental Software Engineering Issues: Critical Assessment and Future Directions, Proc. of International Workshop Dagstuhl Castle*, Germany, September 14-18, 1992, Lecture Notes in Computer Science 706 Springer.

RDF (Resource Description Framework). 1999. *Model and Syntax Specification*. W3C Recommendation. 22-02-1999. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/

Rohl, M. and A.M. Uhrmacher. 2004. "Controlled Experimentation with Agents - Models and Implementations" In *Proc. of 5th Int'l Workshop Engineering Societies in the Agents World*, Toulouse (France), Oct 20-22.

Sarjoughian, H.S.; B.P. Zeigler, and S.B. Hall. 2001. "A Layered Modeling and Simulation Architecture for Agent-based System Development". *Proceedings of the IEEE*, 89 (2), pp. 201-213.

Sierra, C.; J. A. Rodríguez-Aguilar, P. Noriega, M. Esteva, and J.L. Arcos. 2004. *Engineering Multi-agent Systems as Electronic Institutions*. Novática, 170.

Uhrmacher, A.M. 2002. "Simulation for Agent-Oriented Software Engineering". In *Proc. of 1st Int'l Conference on Grand Challenges for Modeling and Simulation*, San Antonio (TX), USA, Jan 27-31.

Wang, Y.; K-L. Tan; and J. Ren. 2002. "A Study of Building Internet Marketplaces on the Basis of Mobile Agents for Parallel Processing". *World Wide Web: Internet and Web Information Systems*, 5(1), pp. 41-66.

Wooldridge, M. 2002. *An Introduction to MultiAgent Systems*. John Wiley & Sons Ltd.

Zambonelli F. and A. Omicini. 2004. "Challenges and Research Directions in Agent Oriented Software Engineering," *Autonomous Agents and Multi-Agent Systems*, 9(3), pp. 253-284.