ELDAMeth Design Process

Giancarlo Fortino, Francesco Rango, and Wilma Russo

Abstract
In this paper the design process documentation template defined in the context of the IEEE FIPA DPDF Working Group (FIPA Design Process Documentation Template, http://www.pa.icar.cnr.it/cossentino/fipa-dpdf-wg/docs.htm, accessed June 2012) is exploited to describe the ELDAMeth agent-oriented methodology. ELDAMeth, which is based on the ELDA agent model and related frameworks and tools, fully supports the development of distributed agent systems and has been applied both stand-alone and in conjunction with other agent-oriented methodologies to different application domains: e-Commerce, information retrieval, conference management systems, content delivery networks, and wireless sensor networks.

1 Introduction

ELDAMeth [7] is an agent-oriented methodology specifically designed for the simulation-based prototyping of distributed agent systems (DAS). It is centered on the ELDA (Event-driven Lightweight Distilled StateCharts Agent) model [9, 12] and on an iterative development process covering DAS Modeling, simulation, and implementation for a target agent platform (currently JADE [2]) and exploits specifically defined frameworks and CASE tools. In particular, the ELDA model is based on three main concepts which are important for enabling dynamic and distributed computation [15, 16]: (1) lightweight agent architecture and agent behaviors driven by events that trigger reactive and proactive computation; (2) agent interaction and cooperation based on multiple coordination spaces that are exploited by

G. Fortino (✉) • F. Rango • W. Russo
Department of Informatics, Modeling, Electronics and Systems (DIMES), University of Calabria, Via P. Bucci, Cubo 41C, 87036 Rende (CS), Italy
e-mail: g.fortino@unical.it; frango@deis.unical.it; w.russo@unical.it

M. Cossentino et al. (eds.), *Handbook on Agent-Oriented Design Processes*,
DOI 10.1007/978-3-642-39975-6_5,

the agents at run-time; (3) coarse-grained strong mobility through which agents can migrate across agent locations by transparently retaining their execution state [6].

Moreover, ELDAMeth can be used either stand-alone, according to the ELDAMeth process reported in Fig. 1, or in conjunction/integration with other agent-oriented methodologies which support the analysis and (high-level) design phases.

In particular, ELDAMeth has been integrated with Gaia [11], PASSI [4], and MCP [9] by using a process-driven method engineering approach [3]. Moreover, ELDAMeth (or previously defined models and frameworks that are now in ELDAMeth) was applied in different application domains: e-Commerce [4, 11], distributed information retrieval [7–9, 12, 14], content distribution networks [10], distributed data mining [5], and wireless sensor networks [1].

Useful references for ELDAMeth:

Fortino, G., Russo, W.: ELDAMeth: a methodology for simulation-based prototyping of DAS. Inform. Softw. Technol. **54**, 608–624 (2012)

Fortino, G., Garro, A., Mascillaro, S., Russo, W.: Using event-driven light-weight DSC-based agents for MAS modeling. Int. J. Agent Orient. Softw. Eng. **4**(2) (2010)

Fortino, G., Rango, F., Russo, W.: Engineering multi-agent systems through statecharts-based JADE agents and tools. Trans. Comput. Collect. Intell. LNCS 7270 **VII**, 61–81 (2012)

Fortino, G., Russo, W., Zimeo, E.: A statecharts-based software development process for mobile agents. Inform. Softw. Technol. **46**(13), 907–921 (2004)

Useful references for ELDAMeth integrations and extensions:

Cossentino, M., Fortino, G., Garro, A., Mascillaro, S., Russo, W.: PASSIM: a simulation-based process for the development of multi-agent systems. Int. J. Agent Orient. Softw. Eng. **2**(2), 132–170 (2008)

Fortino, G., Garro, A., Mascillaro, S., Russo, W.: A multi-coordination based process for the design of mobile agent interactions. In: Proceedings of IEEE Symposium on Intelligent Agents (2009)

Fortino, G., Garro, A., Russo, W.: An integrated approach for the development and validation of multi agent systems. Comput. Syst. Sci. Eng. **20**(4), 94–107 (2005)

1.1 The ELDAMeth Process Lifecycle

ELDAMeth is based on the three phases of the iterative process model shown in Fig. 2:

- The *Modeling* phase produces a specification of a Multi-Agent System (ELDA MAS) fully compliant with the ELDA MAS Meta-model [9] (see Sect. 1.2). Moreover, the platform-independent code of the ELDA MAS is generated in this phase.
- The *Simulation* phase produces MAS execution traces and computes performance indices that are evaluated with respect to the functional and non-functional requirements of the MAS under-development. On the basis of such evaluation,

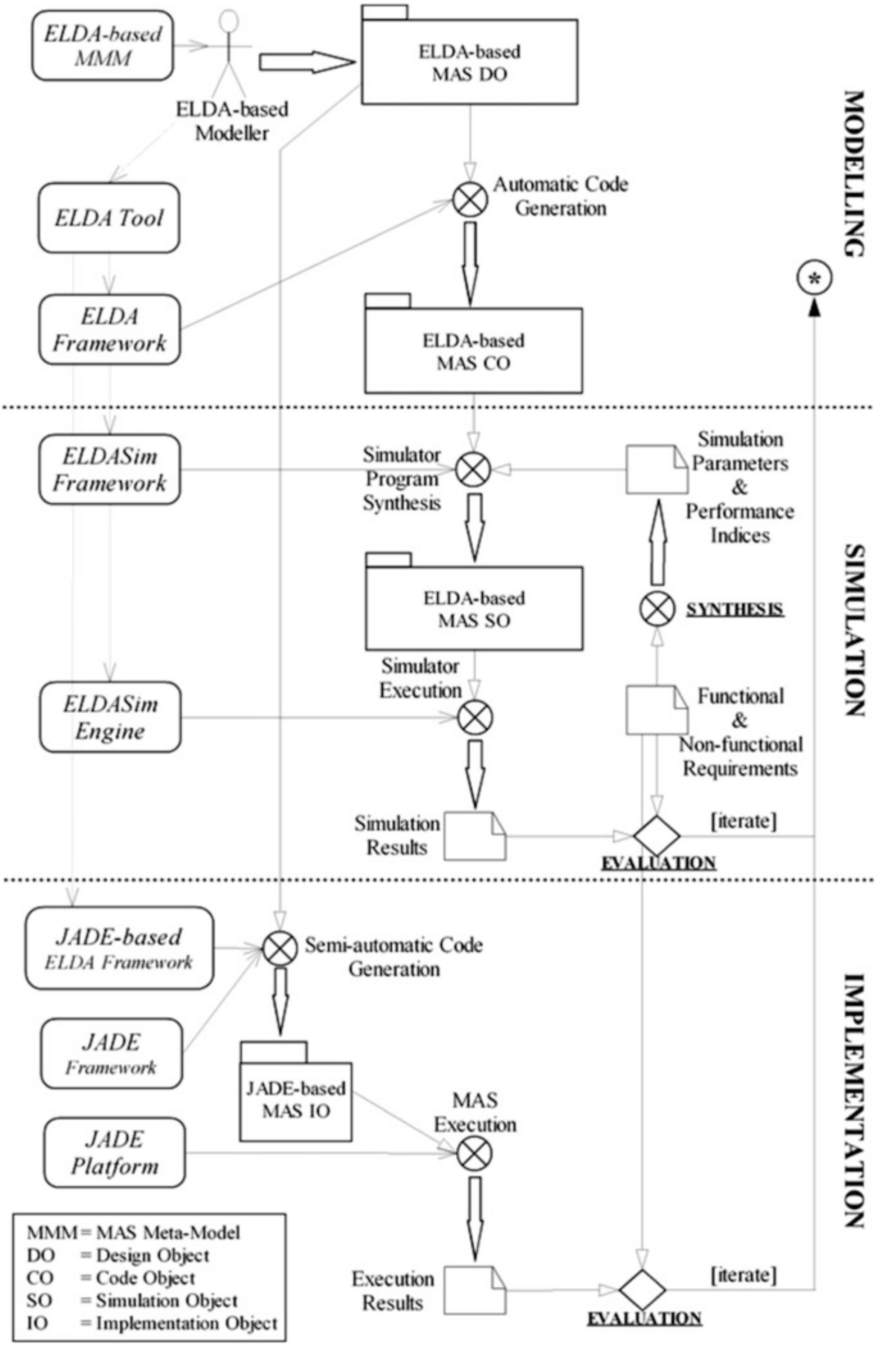

Fig. 1 The traditional ELDAMeth process

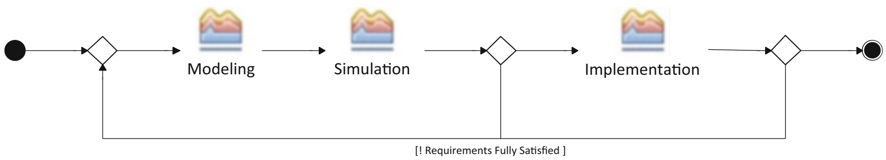

Fig. 2 The ELDAMeth process phases (and iterations)

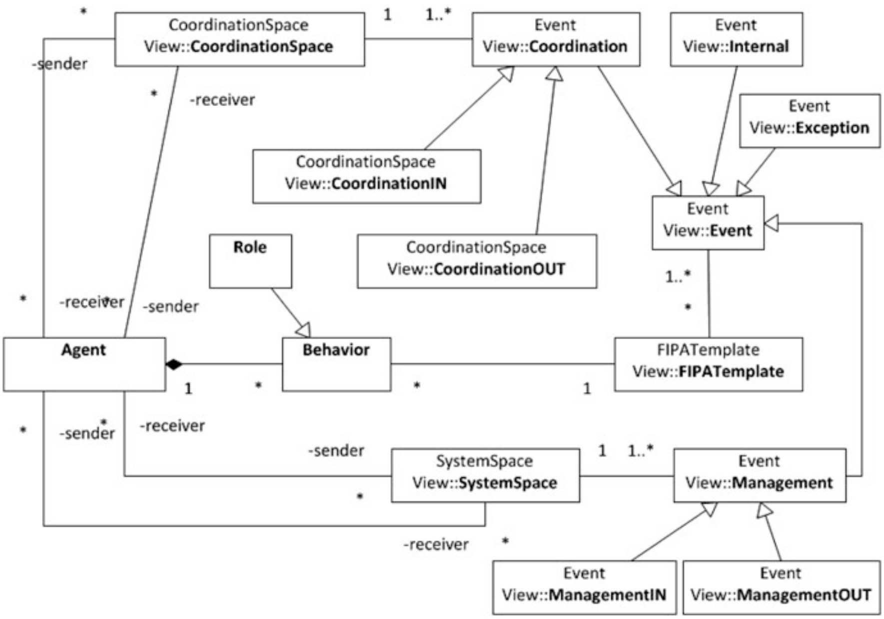

Fig. 3 The ELDA MAS Meta-model

if requirements are satisfied, the *Implementation* phase is carried out; otherwise, the *Modeling* phase is iterated.

- The *Implementation* phase produces the ELDA-based MAS code targeting a specific platform. Currently the JADE platform is exploited [13].

1.2 The ELDA MAS Meta-Model

The MAS Meta-model [9] adopted by ELDAMeth is represented in Fig. 3. The definitions of the MAS Meta-model Elements (MMElements) are reported in Table 1.

Table 1 Definitions of ELDA MAS Meta-model elements

Concept	Definition
Agent	An ELDA agent with multiple behaviors
Role	A role represented by an agent
Behavior	An ELDA agent's behavior is specified through a Distilled StateChart (DSC) [14], which is a hierarchical state machine obtained from Statecharts and based on ECA rules, OR-decomposition, history entrance mechanisms, and UML-like execution semantics based on the run-to-completion step
FIPATemplate	An ELDA behavior is compliant to an extended version of the FIPA agent lifecycle template that allows restoring the agent execution state after agent migration or agent suspension
Event	The interactions of ELDA agents are based on events: – Internal (i.e., self-triggering events) – Management, coordination, and exception (i.e., requests to or notifications from the local agent server). Events can be either OUT-events (generated by the agent and always targeting the local agent server) or IN-events (generated by the local agent server and delivered to target agents)
SystemSpace	SystemSpace provides extensible system services through management (ManagementOUT and ManagementIN) events which allow for agent lifecycle management, timer setting, and resource access
CoordinationSpace	CoordinationSpace provides extensible coordination services through Coordination (CoordinationOUT and CoordinationIN) events which enable coordination acts between agents and between agents and non-agent components (e.g. remote objects, web services) according to specific coordination models. The currently defined inter-agent coordination models are: Direct (synchronous and asynchronous), Tuple-based, and publish/subscribe event-based. The interactions between agent/non-agent components can be based on a general RMI object model or on the Web Services model

2 Phases of the ELDAMeth Process

2.1 The Modeling Phase

The goal of the *Modeling* phase is to provide a detailed design of the MAS under-development in terms of a set of interconnected DSCs [14] representing agent behaviors and/or roles. Figure 4 presents the flow of activities of the Modeling phase. In particular, the two main activities are ELDA Modeling and ELDA Coding. Figure 5 shows the *Modeling* described in terms of activities, roles, and work products. The *Modeling* involves a process role and five work products. The *Modeling* is fully supported by ELDATool, a CASE tool specifically developed to automate modeling, validation and implementation of ELDA-based MAS.

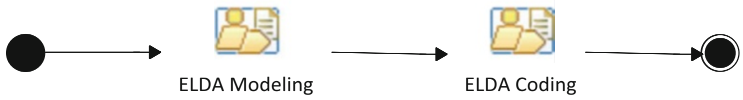

Fig. 4 The *Modeling* flow of activities

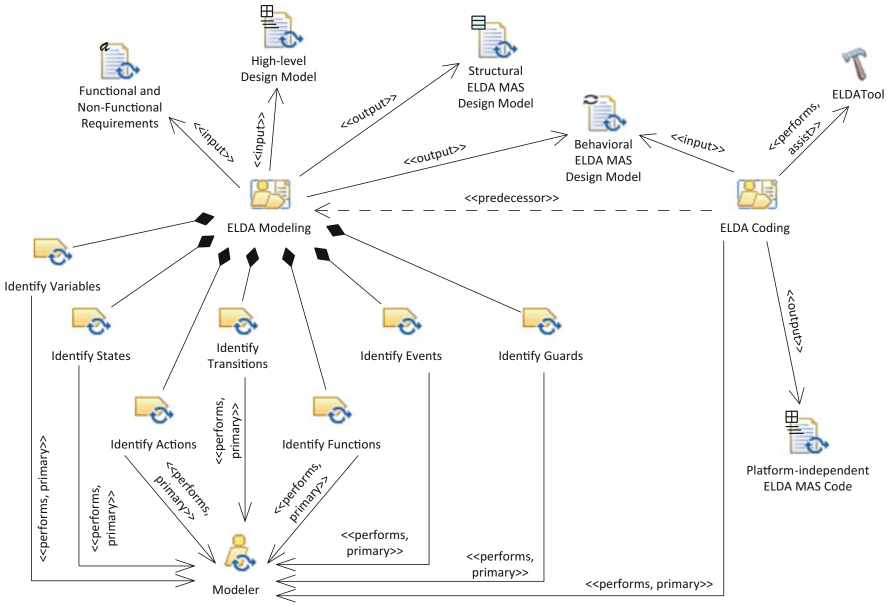

Fig. 5 The *Modeling* phase described in terms of activities, roles, and work products

2.1.1 Process Roles

One role is involved in the Modeling: the Modeler.

Modeler

The Modeler produces a detailed design of the MAS under-development and generates a platform-independent code through the following activities:

- ELDA Modeling: this activity allows the design of the MAS under-development, specifying agent behaviors and/or roles.
- ELDA Coding: the objective of this activity is to generate a platform-independent code for the MAS under-development through ELDATool.

2.1.2 Activity Details

The ELDA Coding activity is an atomic activity that has no tasks and is usually carried out by the Modeler with support of the ELDATool that is able to automatically translate the models produced in the ELDA Modeling activity into

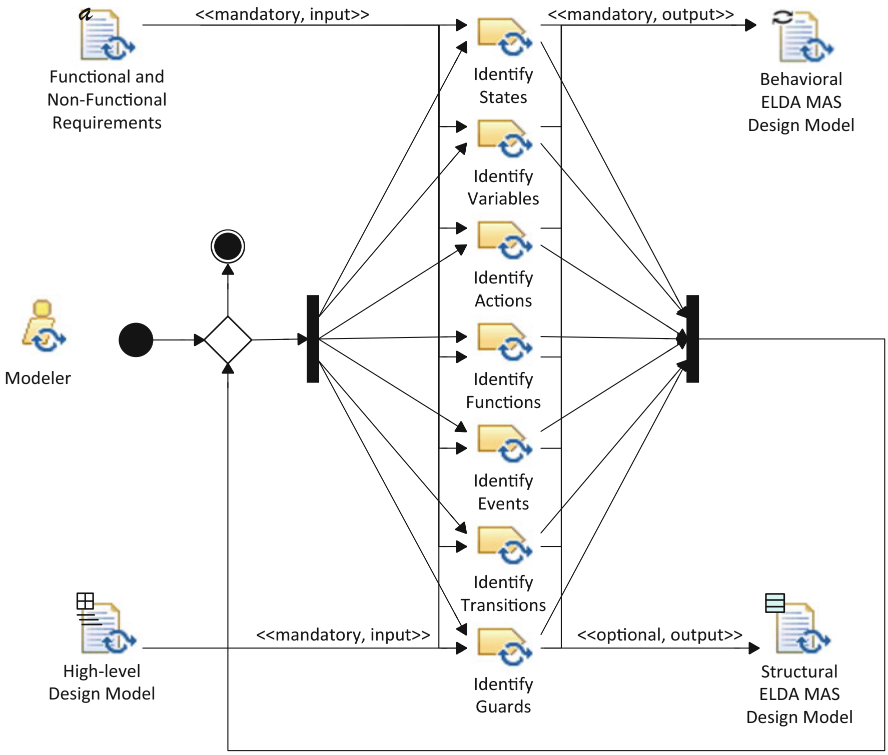

Fig. 6 The flow of tasks of the ELDA Modeling activity

platform-independent code according to the ELDA Framework [9]. Conversely, the ELDA Modeling activity has seven tasks described in the following.

ELDA Modeling Activity

The ELDA Modeling activity is a fundamental activity carried out by the Modeler that produces the behavioral and structural ELDA MAS design based on defined functional and non-functional requirements and on a high-level design model, both deriving from an external system analysis phase not included in ELDAMeth. This activity is composed of seven tasks as shown in Fig. 6; their description is reported in Table 2.

Tasks can be carried out in parallel and iteratively. The mandatory inputs to the *ELDA Modeling* are the *Functional and Non-Functional Requirements document* and the *High-Level Design Model*. The outputs are the (mandatory) *Behavioral ELDA MAS Design Model* and the (optional) *Structural ELDA MAS Design Model*. The former is a set of DSCs, representing agent behaviors and/or roles, whereas the latter is a class diagram representing the interaction relationships among agents and/or roles.

Table 2 Tasks of ELDA Modeling activity

Activity	Task	Task description	Role involved
ELDA Modeling	Identify states	Identification of DSC states	Modeler (perform)
ELDA Modeling	Identify variables	Identification of DSC variables	Modeler (perform)
ELDA Modeling	Identify actions	Identification of DSC actions	Modeler (perform)
ELDA Modeling	Identify functions	Identification of DSC functions	Modeler (perform)
ELDA Modeling	Identify events	Identification of DSC events	Modeler (perform)
ELDA Modeling	Identify transitions	Identification of DSC transitions	Modeler (perform)
ELDA Modeling	Identify guards	Identification of DSC guards	Modeler (perform)

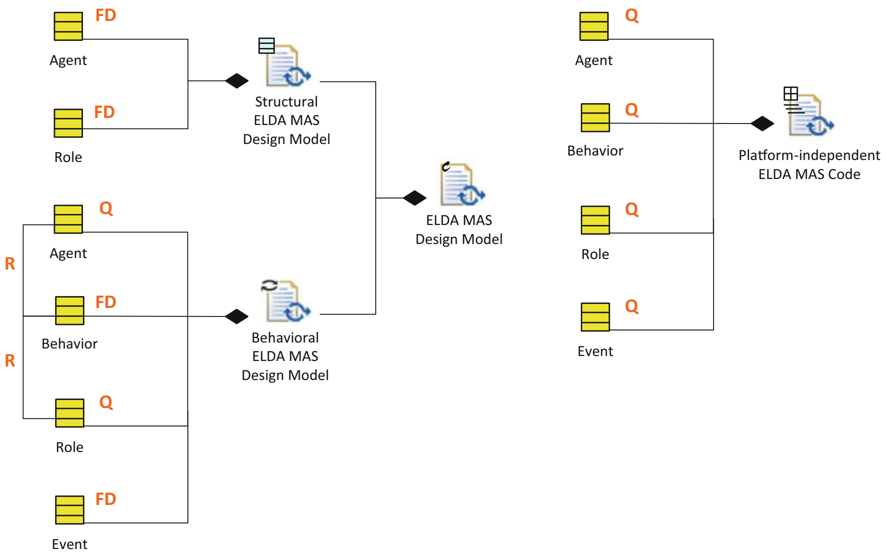

Fig. 7 The *Modeling* work products

2.1.3 Work Products

Figure 7 reports the relationships among the work products of this step (*Modeling* phase) and the ELDA MMMElements (see Sect. 1.2).

Work Product Kinds

Table 3 describes the work products of the *Modeling*.

In the following the *Structural ELDA MAS Design Model*, the *Behavioral ELDA MAS Design Model,* and the *Platform-independent ELDA MAS Code* (specifically the *Reviewer* role code) produced for the CMS case study will be described.

Table 3 Modeling work products kinds

Name	Description	Work product kind
Functional and non-functional requirements	A document defining functional and non-functional requirements of the MAS under-development	Free text
High-level design model	A high-level design model produced by an external method/methodology	Structured
ELDA MAS design model	The detailed design of the MAS under-development	Composite
Structural ELDA MAS design model	The class diagram of the MAS under-development	Structural
Behavioral ELDA MAS design model	The DSC design of the MAS under-development	Behavioral
Platform-independent ELDA MAS code	The platform-independent code generated for the MAS under-development	Structured

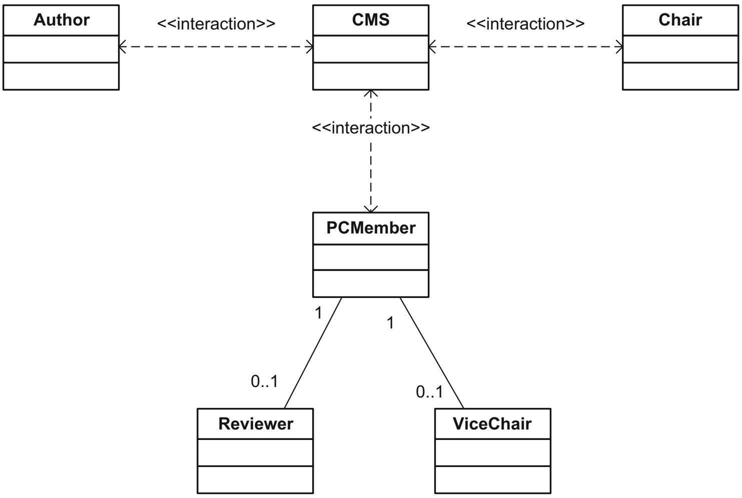

Fig. 8 Class diagram of agents and roles interactions in the CMS case study

Structural ELDA MAS Design Model

In Fig. 8 the *Structural ELDA MAS Design Model* of the CMS case study is portrayed. In particular, five roles are identified: Author, Chair, and PCMember, where a PCMember could be either a Reviewer, or a Vice-Chair, or both. Moreover, CMS is an agent representing the CMS system.

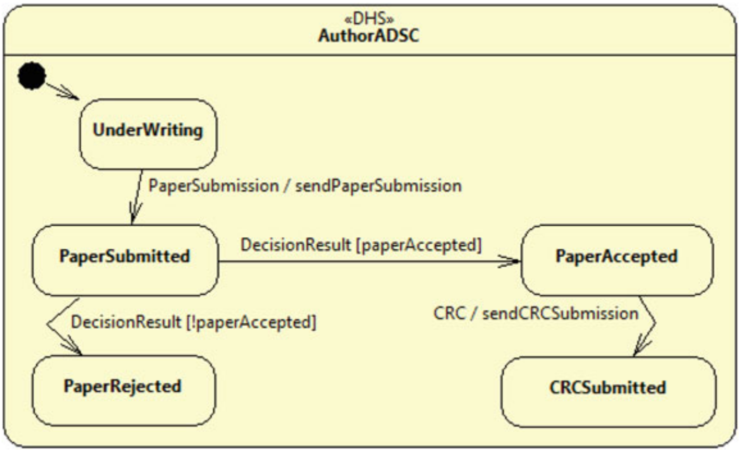

Fig. 9 Author DSC diagram

Table 4 Author actions

Action	Description
sendPaperSubmission	Author submits the paper to the CMS system
sendCRCSubmission	Author submits the CRC to the CMS system

Table 5 Author guards

Guard	Description
paperAccepted	Author checks if the submitted paper has been accepted

Behavioral ELDA MAS Design Model

The *Behavioral ELDA MAS Design Model* of the CMS case study is composed of the DSCs of the five defined roles (Author, Chair, PCMember, Reviewer, and Vice-Chair) and the CMS agent. In the following they are detailed in terms of DSC diagram and event, action and guard tables. The PCMember specification is based on the specifications of Reviewer and Vice-Chair, so a further specification for the PCMember was not defined.

Author

See Fig. 9 and Tables 4, 5 and 6.

Reviewer

See Fig. 10 and Tables 7, 8 and 9.

Table 6 Author events

Event	Sender	Description
PaperSubmission	Author	Internal event sent when Author decides to submit the paper
DecisionResult	CMS	Coordination event containing decision about submitted paper (if it has been accepted or rejected)
CRC	Author	Internal event sent when Author decides to submit the CRC

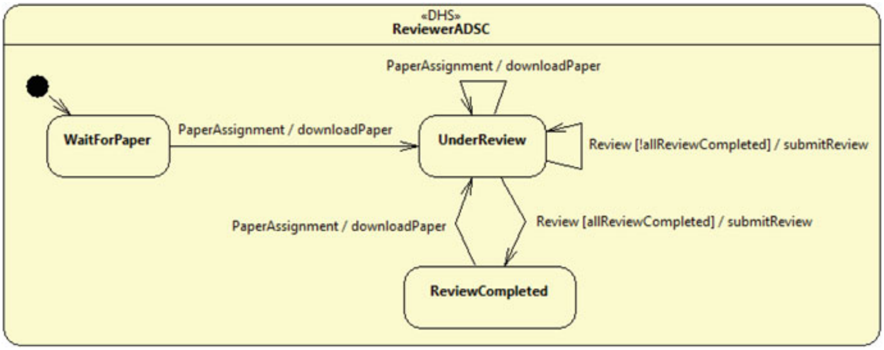

Fig. 10 Reviewer DSC diagram

Table 7 Reviewer actions

Action	Description
downloadPaper	Reviewer downloads papers that have been assigned to it
submitReview	Reviewer sends to the CMS system a review

Table 8 Reviewer guards

Guard	Description
allReviewCompleted	Reviewer checks if all the assigned papers were reviewed

Table 9 Reviewer events

Event	Sender	Description
PaperAssignment	CMS	Coordination event indicating the papers assigned to the Reviewer
Review	Reviewer	Internal event sent when Reviewer completes a review

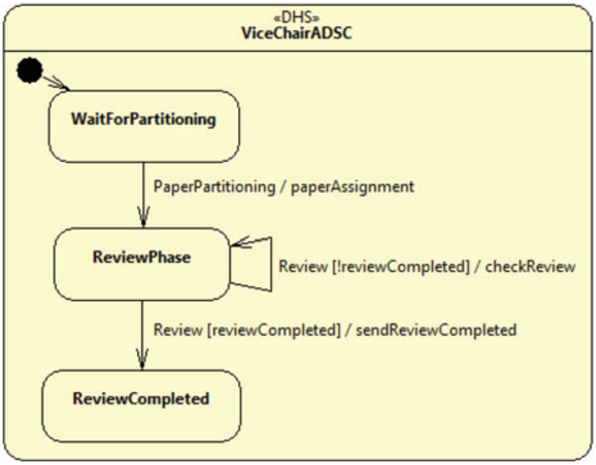

Fig. 11 Vice-Chair DSC diagram

Table 10 Vice-Chair actions

Action	Description
paperAssignment	Vice-Chair sends the event to CMS, indicating the assignment of papers to the reviewers that it manages
checkReview	Vice-Chair checks a review
sendReviewCompleted	Vice-Chair sends the event to CMS, indicating that all the expected reviews (sent from the reviewers managed by this Vice-Chair) were received

Table 11 Vice-Chair guards

Guard	Description
reviewCompleted	Vice-Chair checks if all the expected reviews (sent from the reviewers managed by this Vice-Chair) were received

Table 12 Vice-Chair events

Event	Sender	Description
PaperPartitioning	CMS	Coordination event indicating which reviewers must be managed by Vice-Chair and how to distribute the papers to be reviewed
Review	CMS	Coordination event containing a review

Vice-Chair

See Fig. 11 and Tables 10, 11 and 12.

Chair

See Fig. 12 and Tables 13, 14 and 15.

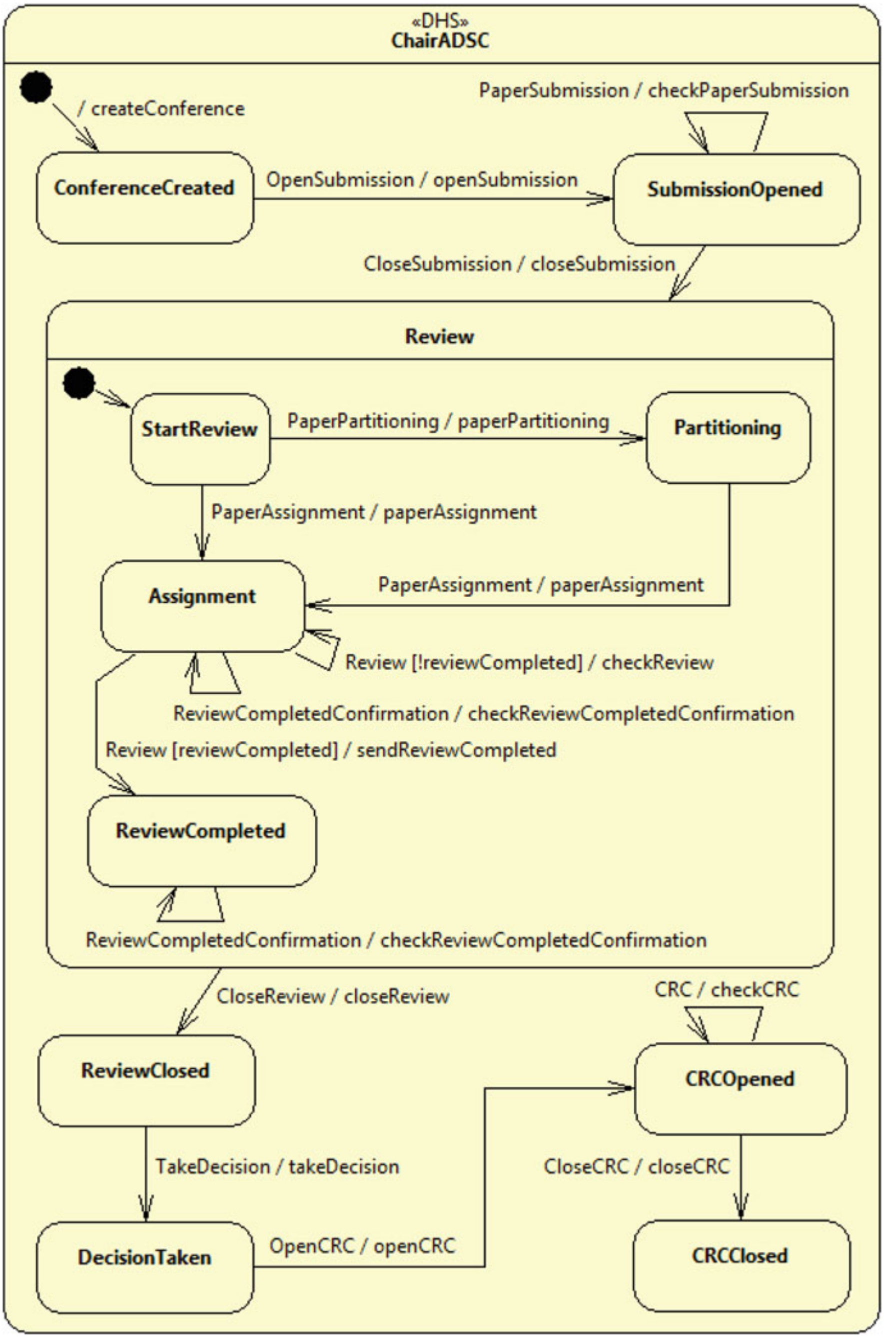

Fig. 12 Chair DSC diagram

Table 13 Chair actions

Action	Description
createConference	Chair creates and initializes the CMS conference system
openSubmission	Chair sends the event to CMS, indicating the opening of the paper submission phase
checkPaperSubmission	Chair checks a submitted paper
closeSubmission	Chair sends the event to CMS, indicating the closure of the paper submission phase
paperPartitioning	Chair sends the event to CMS, indicating how to distribute the papers to be reviewed among the reviewers and how to involve one or more Vice-Chairs in the management of a part of them
paperAssignment	Chair sends the event to CMS, indicating the assignment of some papers to be reviewed to the reviewers that it manages
checkReview	Chair checks a review
checkReviewCompletedConfirmation	Chair checks if all the reviews were successfully received
sendReviewCompleted	Chair sends the event to CMS, indicating that all the expected reviews (sent from the reviewers managed by this Chair) were received
closeReview	Chair sends the event to CMS, indicating the closure of the review phase
takeDecision	Chair sends the event to CMS, indicating the decisions taken on the submitted papers (if they have been accepted or rejected)
openCRC	Chair sends the event to CMS, indicating the opening of the CRC submission phase
checkCRC	Chair checks a submitted CRC
closeCRC	Chair sends the event to CMS, indicating the closure of the CRC submission phase

Table 14 Chair guards

Guard	Description
reviewCompleted	Chair checks if all the expected reviews (sent from the reviewers managed by Chair) were received

CMS

See Fig. 13 and Tables 16 and 17.

Platform-Independent ELDA MAS Code

In Fig. 14 part of the code (variables, actions, guards, and events) of the active behavior of the Reviewer role (see earlier section "Reviewer") produced in the ELDA Coding activity is reported.

Table 15 Chair events

Event	Sender	Description
OpenSubmission	Chair	Internal event sent when Chair decides to open the paper submission phase
PaperSubmission	CMS	Coordination event containing a submitted paper
CloseSubmission	Chair	Internal event sent when Chair decides to close the paper submission phase
PaperPartitioning	Chair	Internal event sent when Chair decides to involve Vice-Chair in the management of the papers to be reviewed
PaperAssignment	Chair	Internal event sent when Chair decides to assign some papers to be reviewed to the reviewers that it manages
Review	CMS	Coordination event containing a review
ReviewCompletedConfirmation	CMS	Coordination event indicating that all the reviews were received
CloseReview	Chair	Internal event sent when Chair decides to close the review phase
TakeDecision	Chair	Internal event sent when Chair wants to start the decision process about the submitted papers
OpenCRC	Chair	Internal event sent when Chair decides to open the CRC submission phase
CRC	CMS	Coordination event containing a submitted CRC
CloseCRC	Chair	Internal event sent when Chair decides to close the CRC submission phase

2.2 The Simulation Phase

The goal of the *Simulation* phase is to support the functional validation and performance evaluation of the MAS model produced in the *Modeling* phase (see Sect. 2.1). Specifically, the ELDASim simulation framework is exploited to fully support such phase. The *Simulation* process is composed of three main activities: *Performance Indices Definition*, *Simulation Implementation*, and *Simulation Execution*, as shown in Fig. 15. *Simulation* specifically involves a process role and five work products, as described in Fig. 16.

2.2.1 Process Roles

One role is involved in the Simulation: the Simulation Designer.

Simulation Designer

The Simulation Designer is responsible for the functional validation and performance evaluation of the MAS under-development through the following activities:

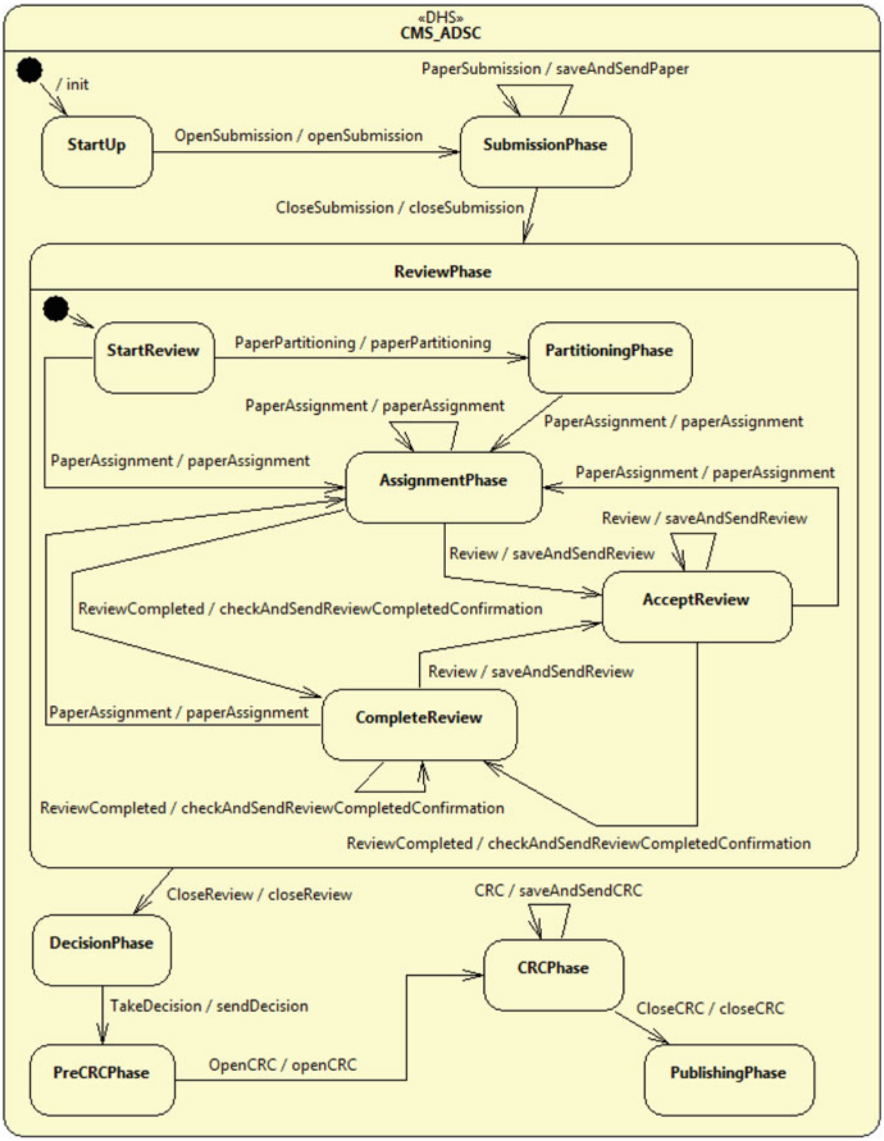

Fig. 13 CMS DSC diagram

- *Performance Indices Definition*: this activity allows the definition of the performance indices which will be evaluated during the simulation.
- *Simulation Implementation*: it produces a simulator program that allows executing the simulation.
- *Simulation Execution*: in this activity the simulation is executed and the simulation results are obtained.

Table 16 CMS actions

Action	Description
init	CMS initializes the conference with parameters decided by Chair at the conference creation and sends the call for paper (CFP) to authors
openSubmission	CMS opens the paper submission phase
saveAndSendPaper	CMS saves a submitted paper and sends a paper submission notification to Chair
closeSubmission	CMS closes the paper submission phase
paperPartitioning	CMS sends the event to Vice-Chair, indicating which reviewers this Vice-Chair must manage and how to distribute the papers to be reviewed among the different reviewers
saveAndSendReview	CMS stores a review and sends it to the corresponding reviewer manager (Chair or Vice-Chair)
saveAndSendCRC	CMS saves and sends a CRC to Chair
closeReview	CMS closes the review phase
sendDecision	CMS sends the decisions, taken by Chair about submitted papers, to authors
closeCRC	CMS closes the CRC submission phase
paperAssignment	CMS assigns some papers to be reviewed to reviewers
openCRC	CMS opens the CRC submission phase
checkAndSendReview CompletedConfirmation	If all the reviewer managers (Chair or Vice-Chair) received all reviews from the reviewers that they manage, CMS sends the event to Chair, indicating that all the reviews have been received

2.2.2 Activity Details

Performance Indices Definition and *Simulation Implementation* are atomic activities that have no tasks, whereas *Simulation Execution* activity has two tasks as described here.

Simulation Execution Activity

The *Simulation Execution* activity comprises the two tasks described in Table 18. The flow of tasks in the *Simulation Execution* activity is reported in Fig. 17.

2.2.3 Work Products

Figure 18 reports the relationships among the work products of this step and the ELDA MMMElements (see Sect. 1.2).

Work Product Kinds

Table 19 describes the work products of the *Simulation*.

Simulator Program

In Fig. 19 the *Simulator Program* template produced for the CMS case study is described.

Table 17 CMS events

Event	Sender	Description
OpenSubmission	Chair	Coordination event indicating the opening of the paper submission phase
PaperSubmission	Author	Coordination event containing a submitted paper
CloseSubmission	Chair	Coordination event indicating the closure of the paper submission phase
PaperPartitioning	Chair	Coordination event sent when Chair decides to involve the Vice-Chair in the management of the papers to be reviewed
PaperAssignment	Chair/Vice-Chair	Coordination event sent by a reviewer manager (Chair or Vice-Chair) to assign some papers to be reviewed to the reviewer that it manages
Review	Reviewer	Coordination event containing a review
ReviewCompleted	Chair/Vice-Chair	Coordination event sent by a reviewer manager (Chair or Vice-Chair) indicating that all the reviews have been received from the reviewers it manages
CloseReview	Chair	Coordination event indicating the closure of the review phase
TakeDecision	Chair	Coordination event sent when Chair wants to decide about submitted papers
OpenCRC	Chair	Coordination event indicating the opening of the CRC submission phase
CRC	Author	Coordination event containing a submitted CRC
CloseCRC	Chair	Coordination event indicating the closure of the CRC submission phase

The methods of the CMS class are:

- `void resetSimulationParams()`: resets the simulation parameters
- `void loadParams(XMLTree configuration)`: loads and initializes the simulation parameters
- `void setupAS()`: performs the setup of the agent servers of the distributed simulated agent platform
- `void createSimPerformanceParamsTabs()`: creates database tables for storing the results obtained from the simulations
- `void setupAndStartCustomSimulation()`: starts the simulation up
- `void setupAgent()`: allows the setup of the agents involvedin the simulation
- `void setAgentCodeDimension()`: sets the code dimension of the agents
- `void startAgent()`: starts the agents up
- `void traceSimPerformanceParams()`: traces the simulation performance parameter values obtained from the simulation
- `void clearAS()`: clears the agent servers up
- `void resetSimPerformanceParams()`: resets the tracing of the simulation results.

STATE	VARIABLES
ReviewerADSC	int reviewCount ELDAId cms

ACTION	CODE

downloadPaper	PaperAssignment evt = (PaperAssignment) e;
	String paperCode = (String) evt.getData();
	download(paperCode);
	reviewCount++;

submitReview	Object review = ((Review) e).getData(); generate(new ELDAEventMSGRequest(self(), new Review(self(), cms, review))); reviewCount--;

GUARD	CODE
allReviewCompleted	return reviewCount == 0;

EVENT	SENDER	TYPE
PaperAssignment	CMS	ELDAEventMSG
Review	Reviewer	ELDAEventInternal

Fig. 14 ELDAFramework-based code of the *Reviewer* role

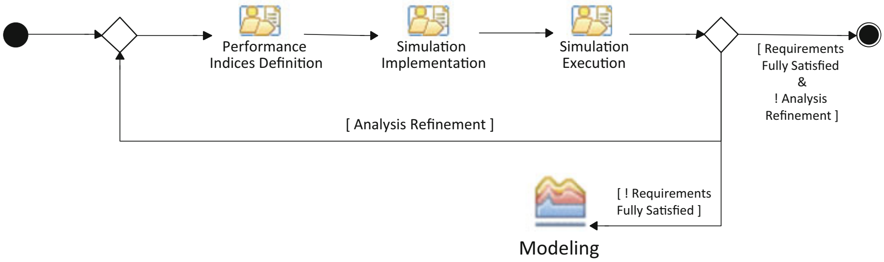

Fig. 15 The Simulation flow of activities

2.3 The Implementation Phase

The goal of the *Implementation* phase is to translate the MAS design model into code for a target platform. In particular, the translation is semi-automatic, supported by the ELDATool, and targeting the JADE platform. The *Implementation* process is composed of two main activities (*Platform-specific ELDA Implementation* and *Testing*), as shown in Fig. 20. In particular, *Implementation* involves two process roles and five work products (see Fig. 21).

2.3.1 Process Roles

Two roles are involved in the *Implementation*: Developer and Tester.

Developer

The Developer is responsible for:

- Platform-specific ELDA Implementation—this activity translates the MAS design model into code generated according to a real target platform (e.g., JADE) through ELDATool.

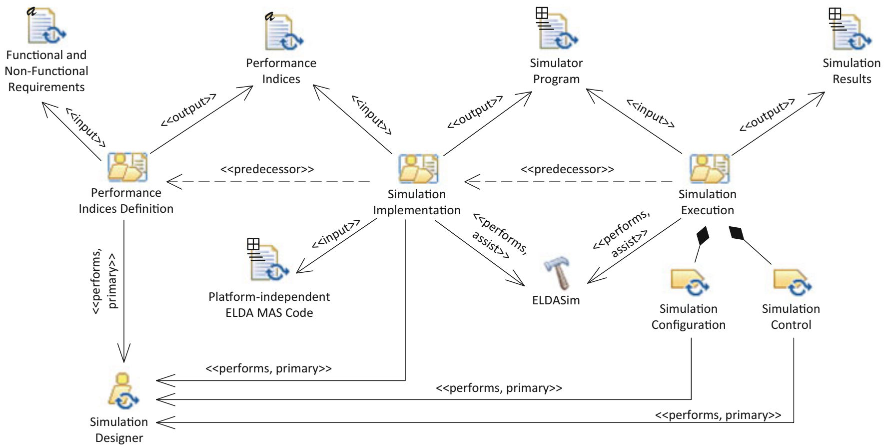

Fig. 16 The Simulation phase described in terms of activities, roles, and work products

Table 18 Tasks of Simulation Execution activity

Activity	Task	Task description	Role involved
Simulation Execution	Simulation Configuration	Configuration of simulation parameters	Simulation Designer (perform)
Simulation Execution	Simulation Control	Control of Simulation Execution	Simulation Designer (perform)

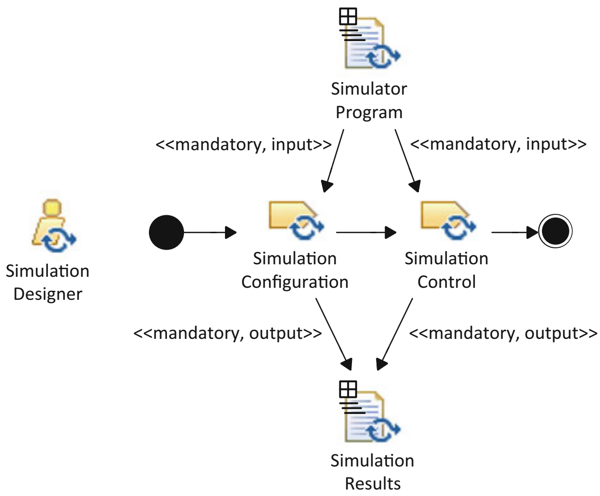

Fig. 17 The flow of tasks of the Simulation Execution activity

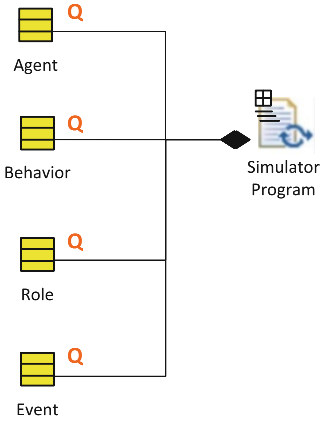

Fig. 18 The Simulation work products

Table 19 Simulation work products kinds

Name	Description	Work product kind
Functional and non-functional requirements	A document defining functional and non-functional requirements of the MAS under-development	Free text
Performance indices	The definition of the performance indices which will be evaluated during the simulation	Free text
Simulator Program	The resulting simulator program that allows executing the simulation	Structured
Platform-independent ELDA MAS code	Platform-independent code generated for the MAS under-development	Structured
Simulation results	Results of executed simulation	Structured

Tester

The Tester is responsible for:

- Testing—this activity executes some tests on the MAS under-development considering the performance indices evaluated during the simulation and produces a document containing the test results.

2.3.2 Activity Details

Platform-specific ELDA implementation and testing activities are atomic and do not have any tasks.

2.3.3 Work Products

The work products produced in this phase are the platform-specific ELDA MAS Code, which is the code of the MAS under-development for the JADE platform, and Testing Results, which is a set of real execution traces and table/plots of computed performance indices.

```
public class CMS extends MASSimulation {

  private static int nReviewerAgent;
  private static int nAuthorAgent;
  private static int nChairAgent;
  private static int nViceChairAgent;

  private static Hashtable<String, SimConfig> simsConfiguration;

  protected void resetSimulationParams(){}
  protected void loadParams(XMLTree configuration) throws Exception{}
  private static void initializeSimsConfiguration(Vector<XMLNode>
                    simsCfg) throws ClassNotFoundException
                  InvalidNodeException, InvalidAttributeException{}
  protected void setupAS(){}
  protected void createSimPerformanceParamsTabs() throws Exception{}
  protected void setupAndStartCustomSimulation() throws Exception{}
  protected void setupAgent() throws Exception{}
  protected void setAgentCodeDimension(){}
  protected void startAgent(){}
  protected void traceSimPerformanceParams() throws Exception{}
  protected void clearAS(){}
  protected void resetSimPerformanceParams(){}
}
```

Fig. 19 Code template of the Simulator Program

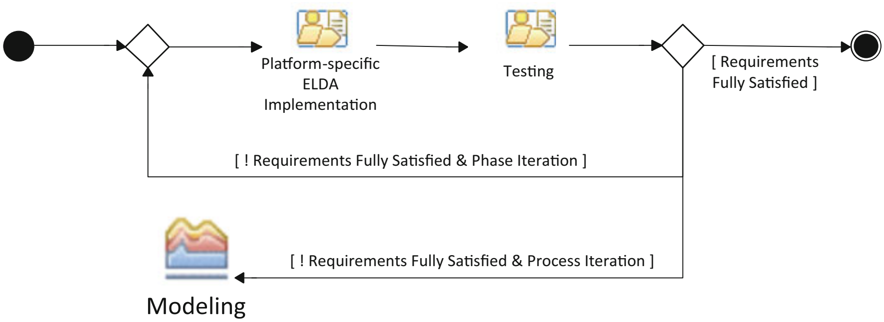

Fig. 20 The Implementation flow of activities

Work Product Kinds

Table 20 describes the work products of the Implementation:

Platform-Specific ELDA MAS Code

In Fig. 22 part of the JADE-based code (variables, actions, guards, and events) of the *Reviewer role* of the *Platform-specific ELDA MAS Code* produced for the CMS case study will be described.

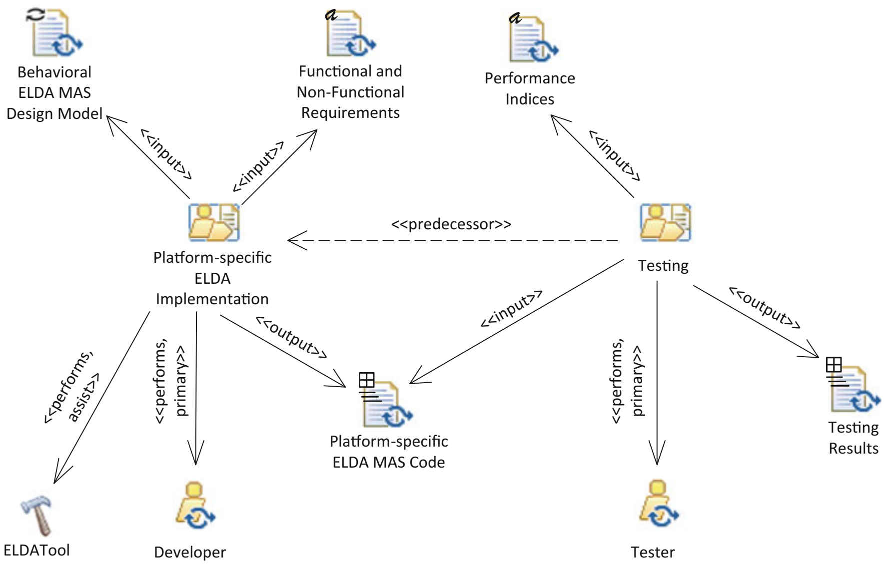

Fig. 21 The Implementation phase described in terms of activities, roles, and work products

Table 20 Implementation work products kinds

Name	Description	Work product kind
Functional and non-functional requirements	A document defining functional and non-functional requirements of the MAS under-development	Free text
Behavioral ELDA MAS design model	The DSC design of the MAS under-development	Behavioral
Platform-specific ELDA MAS code	The MAS code generated according to a real target platform (e.g. JADE)	Structured
Performance indices	The definition of the performance indices evaluated during the simulation	Free text
Testing results	A document containing the results of executed tests	Structured

3 Work Products Dependencies

The diagram in Fig. 23 depicts the dependencies among the different work products.

STATE	VARIABLES
`ReviewerADSC`	`int reviewCount` `AID cms`

ACTION	CODE

`downloadPaper`	`PaperAssignment evt = (PaperAssignment) e;`
	`String paperCode = (String) evt.getData();`
	`download(paperCode);`
	`reviewCount++;`

`submitReview`	`Serializable review = ((Review) e).getData();`
	`ArrayList<AID> target = new ArrayList<AID>();`
	`target.add(cms);`
	`generate(new ELDAEventMSGRequest(self(), new`
	`        Review(self(), target, review)));`
	`reviewCount--;`

GUARD	CODE
`allReviewCompleted`	`return reviewCount == 0;`

EVENT	SENDER	TYPE
`PaperAssignment`	`CMS`	`ELDAEventMSG`
`Review`	`Reviewer`	`ELDAEventInternal`

Fig. 22 The JADE-based code of the *Reviewer* role

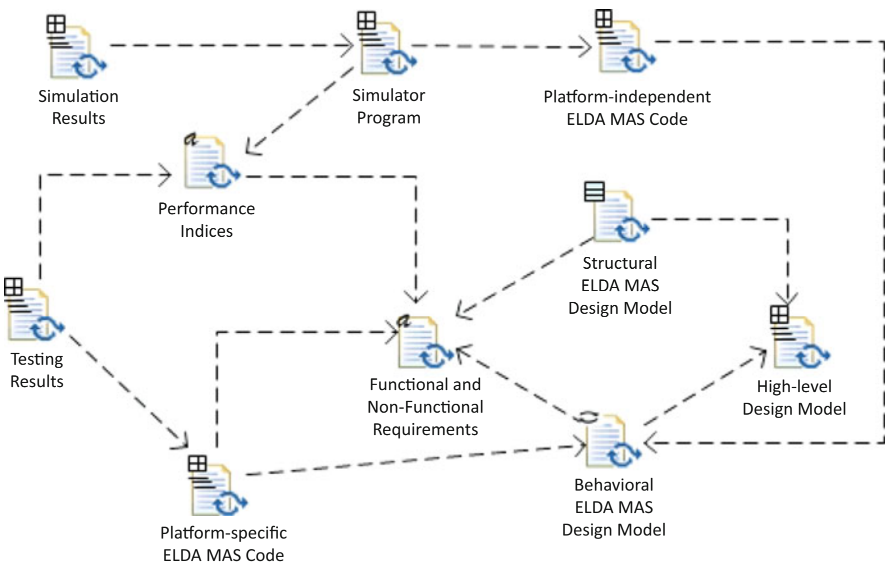

Fig. 23 Work products dependencies diagram

References

1. Aiello, F., Fortino, G., Gravina, R., Guerrieri, A.: A Java-based agent platform for programming wireless sensor networks. Comput. J. **54**(3), 439–454 (2011)
2. Bellifemine, F., Poggi, A., Rimassa, G.: Developing multi-agent systems with a FIPA-compliant agent framework. Softw. Pract. Exper. **31**(2), 103–128 (2001)
3. Brinkkemper, S., Lyytinen, K., Welke, R.: Method engineering: principles of method construction and tool support. In: Proceedings of the IFIP TC8 WG8.1/8.2 Working Conference on Method Engineering (1996)
4. Cossentino, M., Fortino, G., Garro, A., Mascillaro, S., Russo, W.: PASSIM: a simulation-based process for the development of multi-agent systems. Int. J. Agent Orient. Softw. Eng. **2**(2), 132–170 (2008)
5. Di Fatta, G., Fortino, G.: A customizable multi-agent system for distributed data mining. In: Proceedings of the 22nd Annual ACM Symposium on Applied Computing (2007)
6. Fortino, G., Rango, F.: An application-level technique based on recursive hierarchical state machines for agent execution state capture. Sci. Comput. Program. **78**(6), 725–746 (2013)
7. Fortino, G., Russo, W.: ELDAMeth: a methodology for simulation-based prototyping of distributed agent systems. Inform. Softw. Technol. **54**(6), 608–624 (2012)
8. Fortino, G., Garro, A., Mascillaro, S., Russo, W.: A multi-coordination based process for the design of mobile agent interactions. In: Proceedings of IEEE Symposium on Intelligent Agents (2009)
9. Fortino, G., Garro, A., Mascillaro, S., Russo, W.: Using event-driven lightweight DSC-based agents for MAS modeling. Int. J. Agent Orient. Softw. Eng. **4**(2) (2010)
10. Fortino, G., Garro, A., Mascillaro, S., Russo, W., Vaccaro, M. Distributed architectures for surrogate clustering in CDNs: a simulation-based analysis. In: Proceedings of the 4th International Workshop on the Use of P2P, GRID and Agents for the Development of Content Networks (2009)
11. Fortino, G., Garro, A., Russo, W.: An integrated approach for the development and validation of multi agent systems. Comput. Syst. Sci. Eng. **20**(4), 94–107 (2005)
12. Fortino, G., Frattolillo, F., Russo, W., Zimeo, E.: Mobile active objects for highly dynamic distributed computing. In: Proceedings of IEEE International Parallel and Distributed Processing Symposium (IPDPS) (2002)
13. Fortino, G., Rango, F., Russo, W.: Engineering multi-agent systems through statecharts-based JADE agents and tools. In: Nguyen, N.T. (ed.) Transactions on Computational Collective Intelligence VII. Lecture Notes in Computer Science, vol. 7270, pp. 61–81. Springer, Heidelberg (2012)
14. Fortino, G., Russo, W., Zimeo, E.: A statecharts-based software development process for mobile agents. Inform. Softw. Technol. **46**(13), 907–921 (2004)
15. Luck, M., McBurney, P., Preist, C.: A manifesto for agent technology: towards next generation computing. Auton. Agents Multi-Agent Syst. **9**(3), 203–252 (2004)
16. Omicini, A., Zambonelli, F.: Challenges and research directions in agent-oriented software engineering. Auton. Agents Multi-Agent Syst. **9**(3), 253–283 (2004)