

UML Stereotypes Definition and AUML Notations for ADELFE Methodology with OpenTool

Gauthier Picard

IRIT – Université Paul Sabatier
118, route de Narbonne
31062 Toulouse Cedex - France
(33) 5 61 55 82 94

picard@irit.fr

ABSTRACT

ADELFE¹ is a methodology devoted to software engineering of adaptive multi-agent systems. Adaptive software is used in situations in which either the environment is unpredictable or the system is open; in these cases designers cannot implement a global control of the system and cannot list all situations that the system has to be faced with. To solve this problem ADELFE guarantees that the software is developed according to the AMAS (*Adaptive Multi-Agent System*) theory². This theory, based on self-organizing multi-agent systems, enables to build systems in which agents only pursue a local goal while trying to keep cooperative relations with other agents embedded in the system. ADELFE is linked with OpenTool, a commercialized graphical tool which supports UML notation. The paper focuses on our work on OpenTool to take into account AMAS theory and to help designers to develop this kind of systems. The modifications concern (1) the static aspect, with the definition of nine stereotypes, and (2) the dynamic aspects, with the automatic transformations from Agent Interaction Protocols to state machines. State machines simulate the agent behaviors and enable the test and validation of them.

Keywords

Keywords are your own designated keywords.

1. INTRODUCTION

Nowadays, problems to solve in computer science are becoming more and more complex (like information search on the Internet, mobile robots moving in the real world and so on). Systems able to respond to such problems are open and complex because they are incompletely specified, they are immersed in a dynamical environment and especially it does not exist an *a priori* known algorithm to find a solution. This solution must build itself according to interactions the system will have with its environment during its functioning.

1 ADELFE is a French acronym for "Atelier de Développement de Logiciels à Fonctionnalité Emergente". It is a French RNTL-funded project which partners are: ARTAL Technologies (<http://www.artal.fr>) and TNI-Valiosys (<http://www.tni-valiosys.com>) from the industry and IRIT (<http://www.irit.fr/SMAC>) and L3I (<http://www-l3i.univ-lr.fr>) from the academia. This project was started in December 2000 and will end in September 2003.

2 Further information at <http://www.irit.fr/SMAC>.

That led us to propose a theory called AMAS (*Adaptive Multi-Agent System*) theory [10], based on the use of self-organizing systems. This theory has been successfully applied to many projects: a tool to manage the knowledge required to assist a user during information retrieval training [14], an electronic commerce tool for mediation of services [9], a software tool for adaptive flood forecast [8], adaptive routing of the traffic in a telephone network...

Obtained results led us to promote the use of self-organizing systems based on the AMAS theory and to build a methodology for designing such systems. They are required both to reuse our know-how and to guide an engineer during an application design. In that sense, ADELFE, a toolkit to develop software with emergent functionality, is an operational prototype². ADELFE is not a general methodology; it concerns applications in which self-organization makes the solution emerging from the interactions of its parts. It also gives some hints to the designer to tell him if using the AMAS theory is pertinent to build his application at the analysis phase.

This paper is then structured as follow: section 2 gives an overview of the main concepts used in ADELFE. To take into account these concepts, the UML notation has been extended. Therefore, section 3 enumerates the different stereotypes added to the UML notation while section 4 presents some add-ons to OpenTool© (notably the definition of a new schema type) in order to take into account the adaptive agent concept and to allow designers to verify the coherence of the agent behavior.

2. ADELFE Overview

The ADELFE toolkit enables the development of software with emergent functionality and consists of a notation based on UML and AUML, a design methodology, a platform made up of a graphical design tool called OpenTool© and a library of components that can be used to make the application development easier.

In this section, after a brief presentation of the AMAS theory on which ADELFE is based on, the ADELFE methodology is expounded.

2.1 AMAS Theory

The AMAS theory provides a solution to build complex systems for which classical algorithmic solutions cannot be applied [10]. Concerned systems are open and complex. All the interactions the system may have with its environment cannot be exhaustively enumerated, unpredictable interactions can occur during the

system functioning and the system must adapt itself to these unpredictable events.

Classical approaches to solve problems in such a context cannot be applied. The solution provided by the AMAS theory is then to rid ourselves of the global searched goal by building artificial systems for which the observed collective activity is not described in any agent composing it. Each internal part of the system (agent) only pursues an individual objective and interacts with agents it knows by respecting cooperative techniques which lead to avoid cooperation failures (like conflict, concurrency...), called Non Cooperative Situations (NCS). Faced with a NCS, a cooperative agent acts to come back to a cooperative state and permanently adapts itself to unpredictable situations while learning on others. Interactions between agents depend on their local view and on their ability to "cooperate" with each other. Changing these local interactions reorganizes the system and thus changes its global behaviour.

Applying AMAS theory consists in enumerating, according to the current problem to solve, all the cooperation failures that can appear during the system functioning and then defining the actions the system must apply to come back to a cooperative state.

2.2 ADELFE Methodology

The objective of ADELFE is to cover all the phases of a classical software design from the requirements to the deployment. It is based on the RUP (Rational Unified Process) [12], uses UML (Unified Modelling Language) and AUML (Agent-UML) [15][16] notations and adds some specific steps to design adaptive systems. The aim is not to add another methodology but to work on some aspects not already considered by existing ones such as complex environment, dynamic or software adaptation. ADELFE is based on the view of a multi-agent system as a dynamic organization consisting of various cooperative agents. OMG's SPEM (Software Process Engineering Metamodel) [17] has been used to express the ADELFE process. The SPEM vocabulary (WorkDefinitions (WD_i), Activities (A_j) and Steps (S_k)) will be used to expound the methodology.

Only the requirements, analysis and design WorkDefinitions require modifications in order to be tailored to AMAS and are presented in the next paragraphs. Other WorkDefinitions appearing in the RUP remain the same. Any Activity and/or Step added to the RUP is marked with a bold font in the following description tables.

2.2.1 Preliminary and Final Requirements

Contrary to classical approaches, the environment of the system is central in the AMAS theory because the adaptation process depends on the interactions between the system and its environment. This characteristic has led to the addition of one Activity (A6) and one Step (A7-S2) in the "Final Requirements" WD2. To characterize the environment of the system, designers must think about it and qualify it as being accessible or not, deterministic or not, dynamic or static and discrete or continuous. These terms have been reused from [19] and represent a help to later determine if the AMAS technology is required or not to build the studied system (A11). ADELFE is only interested in "cooperative agents" that enable building AMAS. At this point, designers must also begin to think about the situations that can be "unexpected" or "harmful" for the system because these situations

can lead to NCS at the agent level. Therefore, the determination of use cases has been modified to take this aspect into account (S2).

<p><i>WD1: Preliminary requirements</i></p> <p>A1: Define user requirements A2: Validate user requirements A3: Define consensual requirements A4: Establish keywords set A5: Extract limits and constraints</p> <p><i>WD2: Final requirements</i></p> <p>A6: Characterize environment S1: Determine entities S2: Define context S3: Characterize environment</p> <p>A7: Determine use cases S1: Draw an inventory of use cases S2: Identify cooperation failures S3: Elaborate sequence diagrams</p> <p>A8: Elaborate UI prototypes A9: Validate UI prototypes</p>
--

2.2.2 Analysis

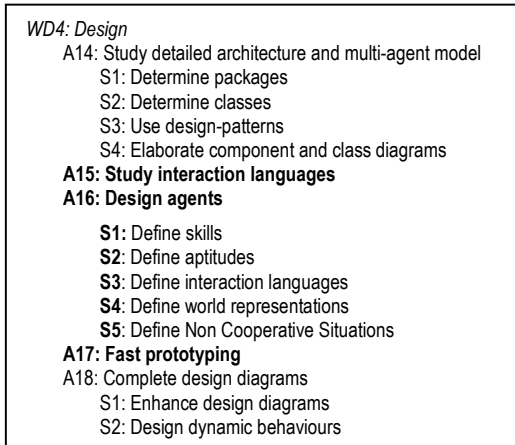
Using of AMAS theory is not a solution necessary to every application. For that reason, ADELFE provides an interactive tool (A11) to help a designer to decide if the use of the AMAS theory is required to implement his application. ADELFE does not assume that all the entities defined during the final requirements are agents. Therefore, this methodology focuses on the agents identification (A12) and some guidelines are then provided to help designers to identify agents [1]. A Step (S3) has also been added concerning the study of agents relationships. Tools enabling the building of such protocols are presented in section 4.

<p><i>WD3: Analysis</i></p> <p>A10: Analyze the domain S1: identify classes S2: Study interclass relationships S3: Construct preliminary class diagrams</p> <p>A11: Verify the AMAS adequacy S1: Verify it at the global level S2: Verify it at the local level</p> <p>A12: Identify agents S1: Study entities in the domain context S2: Identify potentially cooperative entities S3: Determine agents</p> <p>A13: Study interactions between entities S1: Study active/passive entities relationships S2: Study active entities relationships S3: Study agents relationships</p>

2.2.3 Design

Agents being identified and their relationships being studied, designers have now to study the way in which the agents are going to interact (A15) thanks to protocol diagrams (see section 4). ADELFE also provides a model to design cooperative agents (A16). For each type of agents, designers must describe its aptitudes, its interaction language, its world representation and the NCS this agent can encounter. The global function of a self-organizing system is not coded; designers have only to code the local behaviour of the parts composing it. ADELFE provides some generic cooperation failures such as incomprehension, ambiguity, uselessness or conflict. Designers must fill up a table to give the name of each NCS, its generic type, the state in which

the agent must be to detect it, the conditions of its detection and what actions the agent must perform to deal with it. A new Activity (A17) of fast prototyping based on finite state machine has been added to the process (see section 4). It enables designers to verify the behaviour of the built agents.



The aim of the remainder of this paper is to show how the specificities of the AMAS theory have been taken into account to modify and adapt the notations used in the ADELFE methodology.

3. Stereotypes in OpenTool

3.1 Is UML Notation Satisfactory for ADELFE?

In a general way, an agent follows a specific lifecycle: it gets perceptions from its environment and autonomously uses them to decide what to do in order to reach its own goal and, finally, acts to realize the action it has decided before.

However, ADELFE is a methodology which is devoted to a specific kind of agents: cooperative ones. Therefore, even if an agent still follows the same previously defined lifecycle, it has some specific characteristics and is then composed of five parts that will constitute its own behavior:

- Skills that are knowledge about a domain enabling the agent to perform actions.
- Aptitudes which are the abilities an agent possesses to reason on its knowledge (concerning the domain) or on its representation of the world.
- An interaction language which enables the agent to interact and communicate with others in a direct or indirect (possibly, using its environment) way.
- Representations of the world that are knowledge used by an agent to represent itself, other agents or its environment.
- Non Cooperative Situations that an agent must detect and process because these situations are judged "harmful" for both the agent and its viewpoint about the collective.

To enable the developer to deal with these specific components in the ADELFE methodology, some additional notations require to be added in OpenTool.

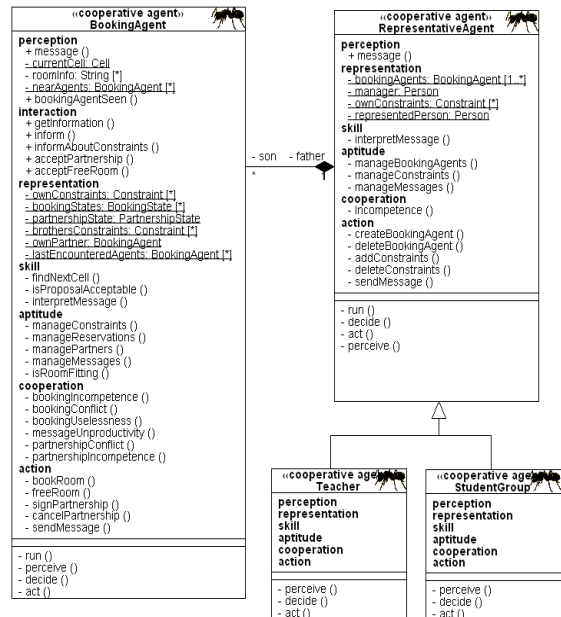


Figure 1. Agent class diagram to illustrate the use of the additional stereotypes.

3.2 How the Notation can be Extended?

Two solutions were available to add these specificities: extending the meta-model or using a UML profile [6]. A meta-model extension is generally used when the concepts concerning a particular domain are well defined and set. However, in the multi-agent domain, the agent concept is not a uniform one, different architectures may exist, cooperative ones for ADELFE, BDI for GAIA [20], etc. Several points of view on multi-agent systems also exist. For instance, ADELFE works on self-organization but other methodologies consider organizations that are known in advance, such as TROPOS [11]. It is therefore more difficult to extend the meta-model and as a result, to "dictate" an agent or multi-agent architecture. That is why the second solution was chosen in order to be more flexible.

So, in ADELFE, nine stereotypes have been defined to express how an agent is formed and/or how its behaviour may be expressed: <<cooperative agent>>, <<characteristic>>, <<perception>>, <<action>>, <<skill>>, <<aptitude>>, <<representation>>, <<interaction>> and <<cooperation>>.

In order to modify the semantics of classes and features depending on the specificities of cooperative agents these stereotypes must be included in the graphical development tool linked with ADELFE. This tool, called OpenTool and commercialised by our partner TNI-Valiosys, supports the UML notation to model applications while assuring that the produced models are valid. Furthermore, some rules are embedded in OpenTool in order to guide designers for correctly using these stereotypes and then implementing agents.

3.3 Stereotypes Description

A table is given to describe each added stereotype by giving its definition, coherency rules that must be followed when using it and an example of use. Examples will be illustrated with a course timetabling problem in which timeslots and locations must be

assigned to teachers and students groups in order to let them meet during lectures. An agent class diagram obtained for the resulting system is shown in figure 1.

Rules that govern the use of each stereotype have been written in the OTScript language which is the action language of OpenTool. All these stereotypes, except <<cooperative agent>>, can be applied to attributes and/or methods.

3.3.1 Stereotype <<cooperative agent>>

Meaning: A <<cooperative agent>> stereotype expresses that an entity is an agent which has a cooperative attitude and can be used to build AMAS. An agent will be implemented using a class that will be stereotyped with <<cooperative agent>>. This class must have a run method that simulates the agent's lifecycle. Therefore, to ensure that this method does exist, an agent inherits from a superclass called CooperativeAgent.

Rules: An agent-stereotyped class inherits (directly or not) from the CooperativeAgent class.

Example: For instance, in the course timetabling application, BookingAgents (BA) have been identified to represent teacher and/or student entities. A BA's goal is to find convenient timeslots in the timetable. A class BookingAgent can then be defined and stereotyped with <<cooperative agent>>. This class inherits from the CooperativeAgent class and therefore contains four methods: run, perceive, decide and act.

3.3.2 Stereotype <<characteristic>>

Meaning: A <<characteristic>> stereotype is used to tag an intrinsic or physical property of a cooperative agent. An attribute represents the value of a property. A method modifies or updates the value of a property. A characteristic can be accessed or called anytime during the lifecycle. It can also be accessed or called by other agents.

Rules: None.

Example: In an application in which robots-agents evolve, it could be the number of wheels that a robot owns, the maximum speed it can reach, the method that enables it to change its maximum speed.

3.3.3 Stereotype <<perception>>

Meaning: A <<perception>> stereotype expresses a means that an agent may use to receive information from the physical or social (other agents) environment. Attributes represent data coming from the environment. Methods are means to update or modify <<perception>>-stereotyped attributes.

Rules: An attribute stereotyped with <<perception>> is necessarily private.

Example: For example, a BookingAgent can get perception about the room it is occupying by managing a private attribute roomInfo. It may store information about BAs that are situated in the same room by using an attribute nearAgents.

3.3.4 Stereotype <<action>>

Meaning: An <<action>> stereotype is used to signal a means for an agent to act on the environment during its action phase. Methods are possible actions for an agent. Attributes are parameters of an action. An agent is the only one that can use its actions.

Rules: An attribute stereotyped with <<action>> is private.

A method that is stereotyped using <<action>> is private and can only be called during the action phase of an agent.

Example: A BookingAgent can act to book an interesting room (using a method called bookRoom()), to free a room because it is no more interesting (method freeRoom()), to send a message to another BA (sendMessage()), etc.

3.3.5 Stereotype <<skill>>

Meaning: A <<skill>> stereotype is used to tag specific knowledge enabling an agent to realize its own partial function. Methods represent reasoning an agent can do. Attributes are data useful to act on the world or parameters of a <<skill>>-stereotyped method. Such an attribute or method can only be accessed/affected or called by the agent itself to express its autonomy of decision. Skills can be represented by a multi-agent system when they need to evolve.

Rules: An attribute or a method that is stereotyped with <<skill>> is necessarily private. Such an attribute can only be used by a <<skill>>-stereotyped method.

A <<skill>>-stereotyped method can only be called during the decision phase of an agent.

Example: For instance, a BookingAgent knows how to interpret a message it received (interpretMessage() method) or evaluate if a received proposal about the booking of a room is interesting (isProposalAcceptable()).

3.3.6 Stereotype <<aptitude>>

Meaning: An <<aptitude>> stereotype expresses the ability of an agent to reason both about knowledge and beliefs it owns. Methods express reasoning that an agent is able to do. Attributes represent functioning data or parameters of a reasoning. A method or an attribute which is stereotyped with <<aptitude>> can only be accessed/affected or called by the agent itself, to express its autonomy of decision.

Rules: An attribute or a method that is stereotyped with <<aptitude>> is necessarily private.

An <<aptitude>> attribute can only be used by a method that is also stereotyped with <<aptitude>>.

A method that is stereotyped with <<aptitude>> can only be called during the decision phase of the agent.

An <<aptitude>>-stereotyped method can only call methods or attributes that are stereotyped with <<perception>>, <<representation>> or <<interaction>>.

Example: A BookingAgent has some constraints concerning the timeslots it can book in the time-table. It must be able to deal with these constraints and to modify or update them. It does this with a method called manageConstraints().

3.3.7 Stereotype <<representation>>

Meaning: A <<representation>> stereotype is a means to indicate world representations that are used by an agent to determine its behavior. Attributes are knowledge units describing an agent. Methods are means to handle representations: access, alteration... Representations that may evolve can be expressed using a multi-agent system.

Rules: An attribute or a method which is stereotyped with <<representation>> is necessarily private.

A <<representation>>-stereotyped attribute can only be used by a method which is stereotyped with <<representation>> or <<aptitude>>.

A <<representation>>-stereotyped method can only be called during the decision phase of the agent.

Example: If it has established a partnership, a BookingAgent knows its partners considering the attribute `ownPartner`. An attribute called `lastEncounteredAgents` enables it to know the identities of the agents it previously encountered on the timetable grid.

3.3.8 Stereotype <<interaction>>

Meaning: An <<interaction>> stereotype tags tools that enable an agent to communicate directly or not with others or with its environment. Methods express the ability an agent owns to interact with others. Attributes represent functioning data or parameters of an interaction. Interactions can be classified into two groups: perceptions and actions which are also tagged with stereotypes (<<perception>> and <<action>>).

Rules: A method stereotyped with <<interaction>> can only call methods stereotyped with <<skill>> or <<interaction>>.

Example: BookingAgents are directly communicating by exchanging messages. Different methods can be stereotyped by <<interaction>> depending on the messages they are dealing with (`inform()`, `acceptPartnership()` methods).

3.3.9 Stereotype <<cooperation>>

Meaning: A <<cooperation>> stereotype expresses that the social attitude of an agent is implemented using rules allowing Non Cooperative Situations (NCS) solving. An agent must have a set of rules (predicates) that enable it to detect NCS. It must also have a method to enable it to solve NCS, this method associates actions with situations in order to process them. A method that is stereotyped with <<cooperation>> is always called during the decision phase of an agent and can be of two kinds:

A method that returns a Boolean result and tries to detect a NCS; its parameters are stereotyped with <<perception>>, <<representation>> or <<skill>>.

A solving method (one per agent a priori) that allows the association of one or several solving actions with each NCS.

Rules: A method stereotyped with <<cooperation>> is private.

Example: One of the possible NCS for a BookingAgent should be "incompetence of partnership". Condition of detection: "A BA meets another BA AND (this latter one is not of the good type OR the cost of the partnership would be equal to or higher than that of former BA)". Actions the former BA must perform to solve the NCS: "memorize the met agent and move in order to meet new BAs". The method that could detect this NCS is named `partnershipIncompetence()` and is stereotyped with <<cooperation>>. A method that could solve this NCS could be, for instance, `freeRoom()` which is stereotyped with <<action>>.

4. From Protocol Diagrams to Finite State Machines in OpenTool

In ADELFE, agents' dynamical behaviours are modelled as classical finite state machines or AIP protocol diagrams. For this

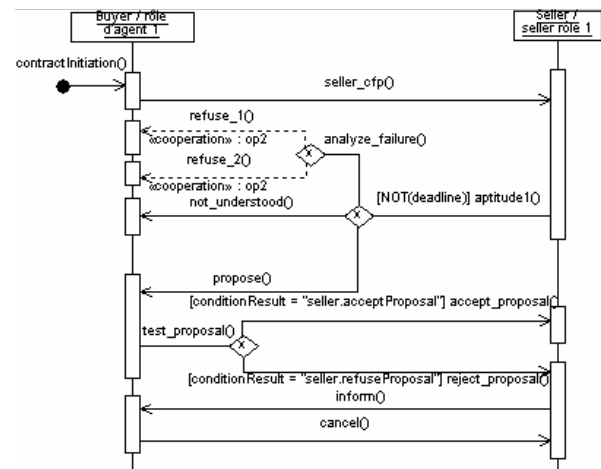


Figure 2. An example of protocol diagram between Buyer and Seller. In this example, there are two classes of agents—Buyer and Seller—and only one role for each class. This diagram has been created with OpenTool.

reason, OpenTool has been enhanced to support AUML notation during the ADELFE project. Since the ADELFE process defines a Fast Prototyping Activity, OpenTool seems to be an adequate tool to simulate agents' behaviours. Actually, the simulation functionality of OpenTool enables designers to simulate objects by running their state machine from an initial collaboration diagram. This simulation enables designers to validate behaviours: it is possible to find if some deadlocks can take place within a protocol, or if some protocols are useless or inconsistent... Thus, the behavior of several agents could be judged conform (or not) to the sequence diagrams described in the analysis work definition.

Until now, OpenTool lacks protocol an automatic process to transform these diagrams into state-machines to simulate them. For this reason, a proposal of transformation of protocol diagrams into state machines is discussed in this section and illustrated by an example protocol taken from [15]. This process has been implemented in OpenTool/ADELFE and is now available for academic evaluation³.

4.1 AUML Protocols in OpenTool

OpenTool/ADELFE proposes a generic protocol diagram as specified in [15] and shown in figure 2. These protocol diagrams are extension of classical UML generic sequence diagrams. The notion of agent roles is added to express a role an agent can have during a protocol, since an agent may have several roles at the same time.

Some decisions have been made to take into account particularities of agents' interactions:

- The graphical representation of time has only sense during activation because AND, OR and XOR branches destroy time continuity; for example, in figure 2, `refuse_1`, `refuse_2` and `not_understood` messages do not occur as a sequence but only one of them is received by the Buyer.
- A protocol diagram must begin with an initial message; in

³ At <http://www.irit.fr/SMAC>

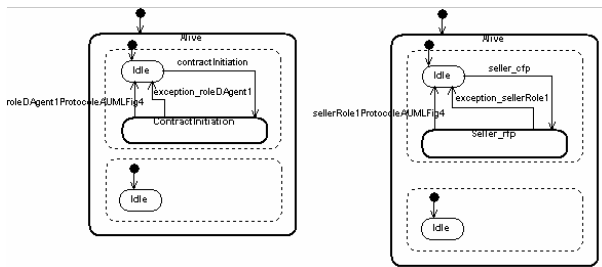


Figure 3. The two top-level state machines corresponding to the two classes appearing in figure 2: the Buyer (left) and the Seller (right). Each class has two roles: a default one (at bottom) and a role one (at top). Dotted lines represent concurrent state-machines in OpenTool.

figure 2, the initial message is contractInitiation;

- The protocol must be represented as a tree from the initial message, with some exceptions.

Moreover, we added some ADELFE-specific notations to fulfil the AMAS technology requirements, such as cooperation:

- An <<interaction>>-stereotyped method must be attached to XOR and OR nodes to specify decision making of an agent; in figure 2, the analyze_failure method is attached to the first XOR node;
- A <<cooperation>>-stereotyped method can be attached to the reception of a message to react to Non Cooperative Situations; in figure 2, the op2 cooperation method is attached to the reception of the refuse_1 message. Such messages are represented by dotted arrows.

Once this notation is embedded in OpenTool/ADELFE, designers can specify dynamical behaviour with protocol diagrams. The next subsection shows how to transform such a protocol diagram into a finite state machine.

4.2 Transformation into Finite State Machines

As previously said, to simulate agents' behaviour, OpenTool requires a dynamic model (Statechart) for each simulated entity (object or agent). Nevertheless, agents' behaviours are modelled as AIP protocol diagrams. By now, there is no formal method to transform a protocol diagram (a particular generic sequence diagram) into a state-chart that OpenTool is able to simulate. Therefore, we propose a method, in terms of the previously expounded choices concerning the semantics of protocol diagrams.

As a preliminary, an initial state-machine is added to each <<cooperative agent>> stereotyped class. This state-machine is composed of an initial state and an Alive state. In the latter state, there are as many concurrent state machines as protocols associated with the agent plus at least one state-machine corresponding to standard behaviour (not conformed to a protocol), as shown in figure 3.

First, a state-machine is associated with each role involved in the protocol. Figure 3 shows the two state machines corresponding to the two classes appearing in the protocol in figure 2. Each class is composed of two concurrent state-machines: a first one corresponding to a role in the protocol (named ContractInitiation role state) and a default one (with a single Idle state).

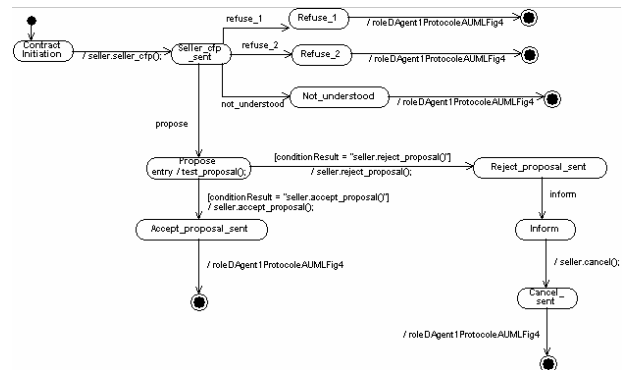


Figure 4. The bottom-level state machine for the Buyer role.

Second, considering the particular semantics previously given to life-line, a sub-state of the role state is associated with each message reception and emission as shown in figure 4. For example, a state is created and associated with the reception of the message seller_cfp, named seller_cfp_sent. Moreover, a state Refuse_1 is added corresponding to the refuse_1 message emission.

Third, a transition between two states is created when a message is received or sent. For example, when the Buyer agent is in the seller_cfp_sent state and receives refuse_1, a transition is created between the seller_cfp_sent and the refuse_1 states. If a state does not receive or send any more messages, a final state is created and both states are linked to return to and Idle state.

Concerning AUML-specific branches, as an <<aptitude>>-stereotyped method is associated with the decision-making process of the branch, a condition⁴ is attached to the transitions in terms of the result of the <<aptitude>> method. To transform XOR branches, the attached <<aptitude>> method returns the signature of the method to activate. This latter is considered as an internal method calling and appears in this "entry" clause of the corresponding state. The signature must be stored in the conditionResult attribute. If the XOR node has n branches, it generates n transitions. In figure 4, conditions corresponding to the XOR branch of the Buyer are [conditionResult = "seller.acceptProposal"] and [conditionResult = "seller.refuseProposal"]. These conditions are attached to transitions to Accept_proposal_sent and Reject_proposal_sent states. A state is created for each AND nodes. To transform them, the transition corresponding to the AND has, as an action to do, all the events of the AND branches. The same message, sent to different instances is done in a single action. Concerning OR branches, the attached aptitude is supposed to manage the choice between the different messages, as for the XOR nodes.

By now, OpenTool/ADELFE automatically generates these state-machines. Designers can attach protocols to agents and then attach new concurrent state machines to simulate the agents' behaviour in order to verify the coherence or to detect deadlocks; but it is not yet automated.

⁴ This condition is expressed in the OTScript language which is a simple action language provided by OpenTool.

5. Conclusion and Perspectives

ADELFE is based on object-oriented methodologies, follows the RUP (*Rational Unified Process*) and uses UML and AUML notations. Some Steps have been added in the classical WorkDefinitions to be specific to adaptive MAS. It is not a general methodology such as GAIA [20] but it has a niche, which concerns applications that require adaptive multi-agent system design using the AMAS theory. ADELFE covers the entire process of software engineering like MESSAGE [7], PASSI [4] and TROPOS [3]. ADELFE is a methodology such as DESIRE [2], MASSIVE [13], INGENIAS/MESSAGE [7], MaSE [5], PASSI [4], PROMETHEUS [18] in which modelling graphical notations are supported by tools. ADELFE differs from other methodologies because it only concerns the development of adaptive multi-agent systems. In this paper, we focus on the static and dynamic aspects added to the graphical tool OpenTool. First, nine stereotypes have been defined to help designers to embed the cooperative agent model. Then, the simulation capability of OpenTool is used to test and validate the agent behaviour. All these adds-on guide designers to use AMAS technique and participate to the development process control. The first prototype is now operational and is currently under evaluation for designing two systems: a flood forecast system and a bioinformatics system.

6. Acknowledgments

We would like to thank the support of the French Ministry of Economy, Finances and Industry as well as our partners: TNI-Valiosys Ltd., ARTAL technologies Ltd. and the IRIT software engineering team.

7. References

- [1] Berton C., Gleizes M-P., Peyruqueou S., and Picard G., ADELFE: a Methodology for Adaptive Multi-Agent Systems Engineering, *Third International Workshop "Engineering Societies in the Agents World" (ESAW-2002)*, 16-17 September 2002, Madrid.
- [2] Brazier F.M., Jonker C. M., and Treur J., Compositional design and reuse of a generic agent model. In *Proceeding of Knowledge Acquisition Workshop - KAW'99*, 1999.
- [3] Castro J., Kolp M., and Mylopoulos J., A Requirements-driven Development Methodology, In *Proceedings of the 13th International Conference on Advanced Information Systems Engineering (CAISE'01)*, Stafford, UK - June, 2001
- [4] Cossentino M., Different Perspectives in Designing Multi-Agent System, *AgeS'02 (Agent Technology and Software Engineering) Workshop at NodE'02*, Erfurt, Germany, October 2002.
- [5] DeLoach S., Analysis and Design Using MaSE and agentTool, *12th Midwest A.I. and Cognitive Science Conference (MAICS01)*, Ohio, 2001.
- [6] Desfray P., UML Profiles Versus Metamodel Extensions: An Ongoing Debate, *OMG's UML Workshops: UML in the .com Enterprise: Modeling CORBA, Components, XML/XMI and Metadata Workshop*, November 2000.
- [7] Eurescom, Project P907-GI - *MESSAGE: Methodology for Engineering Systems of Software Agents*, Deliverable 1 - Initial Methodology, <http://www.eurescom.de/~pub-deliverables/P900-series/P907/D1/P907D1>
- [8] Georgé J-P., Gleizes M-P., Glize P., and Régis C., Real-time Simulation for Flood Forecast: an Adaptive Multi-Agent System STAFF, *Proc. of the AISB'03 symposium on Adaptive Agents and Multi-Agent Systems*, Univ. of Wales, Aberystwyth, 2003.
- [9] Gleizes M-P., Glize P. and Link-Pezet J., An Adaptive Multi-Agent Tool For Electronic Commerce, In *The workshop on Knowledge Media Networking IEEE Ninth International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2000)* 14-16 June 2000 Gaithersburg, Maryland.
- [10] Gleizes M-P., George J-P., and Glize P., A Theory of Complex Adaptive Systems Based on Co-operative Self-Organisation: Demonstration in Electronic Commerce, *Self-Organisation in Multi-Agent Systems (SOMAS)* July 27-28 2000, Milton Keynes UK, A Workshop organised by the Emergent Computing Network.
- [11] Giunchiglia F., Mylopoulos J., and Perini A., The Tropos Software Development Methodology: Processes, Models and Diagrams, *AOSE'02*, Bologna, July 2002.
- [12] Jacobson I., Booch G. and Rumbaugh J., *The Unified Software Development Process*, Addison-Wesley, 1999.
- [13] Lind J., *Iterative Software Engineering for Multiagent Systems: the MASSIVE Method*, LNCS Vol. 1994, 2001.
- [14] Link-Pezet J., Gleizes M-P., and Glize P., FORSIC: a Self-Organizing Training System - *International ICSC Symposium on Multi-Agents and Mobile Agents in Virtual Organizations and E-Commerce (MAMA'2000)* December 11-13, 2000, Wollongong, Australia.
- [15] Odell J., Parunak H.V., and Bauer B., Representing Agent Interaction Protocols in UML, In *Oriented Software Engineering, P. Ciancarini and M. Wooldridge eds.*, Springer-Verlag, Berlin, pp. 121-140, 2001
- [16] Odell J., Parunak H.V., and Bauer B., Extending UML for Agents, In *Proceedings of the Agent Oriented Information Systems (AOIS) Workshop at the 17th National Conference on Artificial Intelligence (AAAI)*, 2000.
- [17] OMG, *Software Process Engineering Metamodel Specification*, <http://cgi.omg.org/docs/formal/02-11-14.pdf>.
- [18] Padgham L., and Winikoff M. - Prometheus : A Pragmatic Methodology for Engineering Intelligent Agents, *Workshop on Agent-Oriented Methodologies at OOPSLA 2002*
- [19] Russel S., and Norvig P., *Artificial Intelligence: A Modern Approach*, Prentice Hall Series, 1995.
- [20] Wooldridge M., Jennings N.R., and Kinny D., A Methodology for Agent-Oriented Analysis and Design, In *Proceedings of the 3rd International Conference on Autonomous Agents (Agents 99)*, pp 69-76, Seattle, WA, May 1999.