Published in B. Henderson-Sellers and P. Giorgini, editors, *Agent Oriented Methodologies*, chapter 7, pages 172-202, Idea Group Publishing, 2005.

Chapter VII

Engineering Adaptive Multi-Agent Systems: The ADELFE Methodology

Carole Bernon
IRIT – University Paul Sabatier, France

Valérie Camps
IRIT – University Paul Sabatier, France

Marie-Pierre Gleizes
IRIT – University Paul Sabatier, France

Gauthier Picard
IRIT – University Paul Sabatier, France

Abstract

This chapter introduces the ADELFE methodology, an agent-oriented methodology dedicated to the design of systems that are complex, open, and not well-specified. The need for its development is justified by the theoretical background given in the first section, which also gives an overview of the concepts on which multi-agent systems developed with ADELFE are based. A methodology is composed of a process, a notation, and tools. Tools are presented in the second section and the process in the third one, using an information system case study to better visualize how to apply this process.

Published in B. Henderson-Sellers and P. Giorgini, editors, *Agent Oriented Methodologies*, chapter 7, pages 172-202, Idea Group Publishing, 2005.

The last part of the chapter assesses strengths and limitations of ADELFE. We note that its main strength is also its main limitation—it is a specialized methodology, especially suited to the development of software with emergent functionalities.

Introduction

Usually, classical design of computational systems requires some important initial knowledge in the sense that the exact purposes of the system and every interaction to which it may be confronted in the future have to be known. However, at the same time, today's problems are becoming more and more complex (e.g., information searching on the Internet, mobile robots moving in the real world). Indeed, systems that are able to deal with such problems are also becoming open and complex; they are immersed in a dynamical environment; they are often incompletely specified and, especially, an a priori known algorithm does not exist to find a solution. Classical approaches then become inadequate and a new way to tackle such problems is necessary.

Our research work, for several years now, has essentially focused on these kinds of systems and has led us to propose *Adaptive Multi-Agent Systems* (AMAS) as an answer (Camps, Gleizes, & Glize, 1998; Capera, Georgé, Gleizes & Glize, 2003; Gleizes, Georgé & Glize, 2000; Piquemal-Baluard, Camps, Gleizes, & Glize, 1996). These systems are composed of agents that permanently try to maintain cooperative interactions with others. We have built, with success, several systems based on the use of adaptive agents in different areas. To ease and promote this kind of programming, we then developed the ADELFE methodology, the aim of which is to help and guide designers when developing AMAS.

The remainder of this section briefly presents the foundation of adaptive multi-agent systems and then explains how to implement adaptation in such systems. After that, the main characteristics of ADELFE, as well as the context of its presentation, are given.

Theoretical Background: Adaptive Multi-Agent Systems

In a general way, when conceiving a system, a designer wants it to realize the right function; the system must be "functionally adequate." But openness and dynamics are sources of unexpected events and an open system plunged into a dynamic environment has to be able to adapt to these changes, to self-organize.

Published in B. Henderson-Sellers and P. Giorgini, editors, *Agent Oriented Methodologies*, chapter 7, pages 172-202, Idea Group Publishing, 2005.

If every component of a system is endowed with the capability to locally rearrange its interactions with others, this ability of self-organization at the lowest level permits changes in the global function without coding this modification at the upper level of the system. Self-organization is a means to make the system adapt but also to overcome complexity. If a system is complex and its algorithm unknown, it is impossible to code its global function. This function has then to emerge at the macro level (the system level) from the interactions at the micro level (component level). Moreover, this global function cannot be known at the component level, and a component just needs some local criteria to rearrange its interactions. A proven theorem on functional adequacy says that "For any functionally adequate system in a given environment, there is a system having a cooperative internal medium which realizes an equivalent function" (Camps et al., 1998, p. 8). In other words, it is sufficient to build a system whose components have a cooperative attitude to make it realize an expected function. Cooperation is the local criterion that enables a component to find the right place within the organization and that ensures that the system taken as a whole is functionally adequate.

Highly relevant to our work in the agent domain, this theory has been mapped onto multi-agent systems[1] giving rise to what we call Adaptive Multi-Agent Systems (AMAS).

Implementation of Self-organization: Cooperative Agents

Any agent in an AMAS follows a specific lifecycle that consists of three steps:

- The agent gets perceptions from its environment;
- It autonomously uses them to decide what to do in order to reach its own goal; and
- It acts to realize the action on which it has previously decided.

More precisely, each agent follows this lifecycle while trying to keep cooperative local interactions with others.

These cooperative agents are equipped with five modules to represent their "physical," "cognitive," or "social" capabilities (Picard, 2003). Each module represents a specific resource for the agent during its "perceive-decide-act" lifecycle. The first four modules are quite classical in an agent model (Brazier, Jonker, & Treur, 1999; Wooldridge, 2002); the novelty comes from the fifth one:

Published in B. Henderson-Sellers and P. Giorgini, editors, *Agent Oriented Methodologies*, chapter 7, pages 172-202, Idea Group Publishing, 2005.

- The skill module represents knowledge on specific fields that enables agents to realize their partial function. No technical constraints are required to design and develop skills. For example, they can be represented as a classical or fuzzy knowledge base of facts and rules on particular domains. They can also be decomposed into an MAS at a lower level to support learning if they need to evolve.
- The representation module enables an agent to create its own representation about itself, other agents, or the environment it perceives. For example, representations can be implemented as a classical or fuzzy knowledge base. As with skills, representations can be decomposed into an MAS when learning capabilities on representations are needed.
- The interaction module is composed of perceptions and actions. Perceptions represent the inputs the agent receives from its environment. Actions represent the outputs and the way the agent can act on its physical environment, its social environment, or itself (considering learning actions, for example). Both perceptions and actions may have different granularities—from simple effectors providing activation for a robot to semantically complex message sending for social agents.
- The aptitude module provides capabilities to reason on perceptions, skills, and representations – for example, to interpret messages. For example, these aptitudes can be implemented as inference engines if skills and representations are coded as knowledge bases.
- The cooperation module embeds local rules to be locally "cooperative." Being cooperative does not mean that an agent is always helping other agents or that it is altruistic, but only that it is able to recognize states that it judges opposed to what it knows as being an "ideal cooperation" (that is to say fulfilling three conditions: all perceived signals are understood, reasoning on them leads to conclusions and these conclusions are useful). These states are called "cooperation failures" or Non Cooperative Situations (NCS). From an observer's viewpoint, the whole system is able to detect any non-cooperative state coming either from the occurrence of novelty or resulting from feedback returned by the environment concerning a previously erroneous response of the system.

This theory has been applied to many projects: foraging ant simulation (Topin et al., 1999), knowledge management, e-commerce (Gleizes, Glize, & Link-Pezet, 2003), flood forecasting (Georgé, Gleizes, Glize, & Régis, 2003), routing in telephonic network, mechanical design, bio-informatics, and so forth. To ease our future work, design tools were required. They were also needed to promote this kind of programming towards designers not accustomed to developing AMAS.

Published in B. Henderson-Sellers and P. Giorgini, editors, *Agent Oriented Methodologies*, chapter 7, pages 172-202, Idea Group Publishing, 2005.

The Methodology ADELFE: Context of Presentation

It was soon recognized that even though several agent-oriented methodologies already existed (Iglesias, Garijo, & Gonzalez, 1998; Wood & DeLoach, 2000), none was suited to handle complexity, dynamics, openness, or software adaptation. This led us to develop a toolkit – ADELFE – to work on some aspects not already considered by existing methodologies and to support the AMAS theory that has been briefly introduced above (Bernon, Camps, Gleizes, & Picard, 2002). ADELFE is an acronym that, translated from French, means "toolkit to develop software with emergent functionality."

In a general way, a methodology is made up of a process, some notations, and tools to support these notations and/or help the developer (Shehory & Sturm, 2001). ADELFE provides a specific process adapted from an interpretation of the Rational Unified Process (RUP) (Kruchten, 2000) according to the Neptune Project (*http://www.neptune.irit.fr*). Some additions have been made to take into account specificities of the AMAS theory, for example, the characterization of the environment of the system, the identification of cooperation failures, and so forth.

In the third section of this chapter, each of these extensions is exemplified relating to a particular case study. The chosen case study consists of designing a system that enables end-users and service providers to get in touch when they share common centres of interest in a dynamic and distributed context (such as the problem described in Gleizes *et al.* [2000]). The main requirement of such an electronic information system is to enable 1) end-users to find relevant information for a given request, and 2) information providers to have their information proposed to relevant end-users.

More precisely, the system has to provide:

- Personalized assistance and notification for the end-users;
- Propagation of requests between the actors of the system;
- Propagation of new information only to potentially interested end-users; and
- Acquisition of information about end-users' real interests, in a general manner, and about the information offers of providers.

We are clearly situated in a system where every end-user and service provider has an individual goal: to answer the request he/she has to solve. Each end-user and service provider does not know the global function of the system. The system is strongly open because a great number of appearances or disappearances of

Published in B. Henderson-Sellers and P. Giorgini, editors, *Agent Oriented Methodologies*, chapter 7, pages 172-202, Idea Group Publishing, 2005.

end-users and/or service providers may occur. Moreover, an algorithmic solution is not known. In this context, classical approaches to tackle such a problem cannot be applied. Using an AMAS in such a context is clearly relevant—we can say that such a system is functionally adequate when a satisfied end-user wants to use services of the system again, and when each service is fully used, namely, in the most profitably way for the supplier.

Although ADELFE is an agent-oriented methodology suited to develop applications based on the AMAS technology, it does not assume that the designer is specialized in this field. Therefore, some additional notations are provided as well as some tools to help or guide the designer throughout the process application. An overview of these different tools is given in the next section. The process of ADELFE is expounded upon in the third section, by using the case study as illustration. Strengths and weaknesses of ADELFE are finally presented, along with some omissions that were intentionally made when defining this methodology.

Tools Linked with ADELFE

To help in its use, ADELFE is based on "standards" such as the RUP and UML; it also uses AUML[2] (Odell, Van Dyke Parunak, & Bauer, 2000) to express agent interaction protocols. However, being based on standards is not sufficient, and tools are also required. This section gives an overview of the three main tools integrated into the ADELFE toolkit:

- A tool that analyzes answers given by the designer to tell him/her if using the AMAS technology is useful to implement the target system;
- OpenTool, a graphical modelling tool that supports the UML notation and that has been modified to integrate new stereotypes and AUML interaction protocols; and
- An interactive tool that describes the process and helps the designer to apply it.

The AMAS Adequacy Tool

Not every designer needs to use the AMAS theory to build a system. Indeed, if the algorithm required to solve the task is already known, if the task is not complex, or if the system is closed and nothing unexpected can occur, this kind

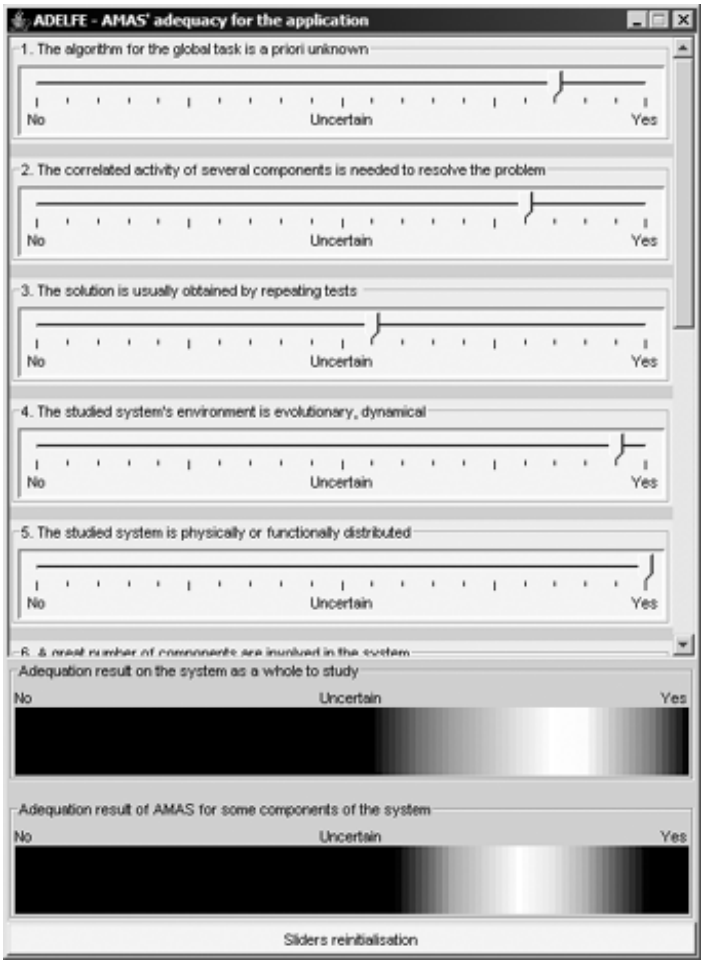
Published in B. Henderson-Sellers and P. Giorgini, editors, *Agent Oriented Methodologies*, chapter 7, pages 172-202, Idea Group Publishing, 2005.

of programming is completely useless and a more adapted one can then be considered. Thus, ADELFE gives the designer a tool to study the adequacy of the AMAS technology more easily.

This adequacy is studied at two levels: the global one (system) and the local one (components). At the system level, eight criteria are studied:

1. Is the global task incompletely specified? Is an algorithm a priori unknown?
2. Is the correlated activity of several entities needed to solve the problem?
3. Is the solution generally obtained by repetitive tests? Are different attempts required before finding a solution?
4. Can the system environment evolve? Is it dynamic?
5. Is the system functionally or physically distributed? Are several physically distributed components needed to solve the global task? Or is a conceptual distribution needed?

Figure 1. Overview of the AMAS adequacy tool



Published in B. Henderson-Sellers and P. Giorgini, editors, *Agent Oriented Methodologies*, chapter 7, pages 172-202, Idea Group Publishing, 2005.

6. Does a great number of components needed?
7. Is the studied system non-linear?
8. Finally, is the system evolutionary or open? Can new components appear or disappear dynamically?

And at the component level, three more criteria are used:

9. Does a component have only a limited rationality?
10. Is a component "big" or not? Is it able to do many actions, to reason a lot? Does it need significant abilities to perform its own task?
11. Can the behaviour of a component evolve? Does it need to adapt to the changes of its environment?

These questions are asked of designers using a graphical interface as visualized in Figure 1. A designer uses a slider to answer a question by giving a rate among 20 possibilities ranging from "yes" to "no." His/her answers are then analyzed by the support decision tool. The two areas at the bottom of the graphical tool window show the answers of ADELFE regarding the global level and the local one. By clicking on those areas, an interpretation of the results can be obtained.

OpenTool Modified for ADELFE

OpenTool is a graphical modelling tool supporting the UML notation; it is developed and commercialized by our project partner, TNI-Valiosys, and is embedded in the ADELFE toolkit. This tool permits applications modelling while assuring that the produced models are valid.

On the one hand, some deficiencies exist in the UML notation for dealing with the specific modules composing a cooperative agent. On the other hand, AUML diagrams to model interaction protocols between agents are needed and must be supported by the tools linked with ADELFE. OpenTool has thus been modified to allow expression of cooperation failures, to deal with the components of an agent that constitute its behaviour, and to embed some AUML diagrams.

Expressing the Behaviour of an Agent through Stereotypes

Two solutions were available to express the behaviour of agents: extending the meta-model or using a UML profile (Desfray, 2000). The former solution has not

Published in B. Henderson-Sellers and P. Giorgini, editors, *Agent Oriented Methodologies*, chapter 7, pages 172-202, Idea Group Publishing, 2005.

been chosen because concepts concerning the agent domain or the multi-agent one are not so well defined and set; many points of view still exist. It is therefore difficult to "dictate" an agent or a multi-agent architecture.

We thus chose the latter solution by defining nine stereotypes to show how an agent is formed and/or how its behaviour is expressed. They are defined and embedded into OpenTool and rules (written in the OTScript language linked with OpenTool) are given to govern their use:

- The first stereotype, «cooperative agent», expresses that an entity is an agent that has a cooperative attitude and can be used to build AMAS. An agent is implemented using a class stereotyped with «cooperative agent» that must have methods (perceive, decide, act) that simulate the agent's lifecycle.
- As introduced in the previous section, modules are related to a cooperative agent and stereotypes have been associated with each one of these modules: «skill», «aptitude», «representation», «interaction», «perception», «actions» (perceptions and actions are specific interactions), and «cooperation».
- The ninth stereotype, «characteristic», is used to tag an intrinsic or physical property of a cooperative agent (for example, the address of a service provider). A characteristic can be accessed or called anytime during the lifecycle. It can also be accessed or called by other agents (for example, if an end-user wants to know the address of a service provider).

A class called CooperativeAgent is the base class of all these stereotypes (see the third section, devoted to the process of ADELFE). The «cooperative agent» stereotype can only be applied to a class inheriting from that one. The last eight stereotypes can be applied to attributes and/or methods of this class. Attributes correspond to the data manipulated in the modules composing a cooperative agent; methods are means to access or act on these attributes.

More details about those stereotypes and the rules linked with them (e.g., an agent inherits from a super-class called CooperativeAgent, methods stereotyped with «cooperation» are always called during the decision phase of an agent, etc.) can be found in Picard (2003).

Integrating the AUML Notation

Agents' interactions and languages are specified by using the AUML interaction protocol model (Odell, Van Dyke Parunak, & Bauer, 2001). To fit with AMAS

Published in B. Henderson-Sellers and P. Giorgini, editors, *Agent Oriented Methodologies*, chapter 7, pages 172-202, Idea Group Publishing, 2005.

specificities, the model has been extended and included in OpenTool functionalities (Bernon et al., 2004).

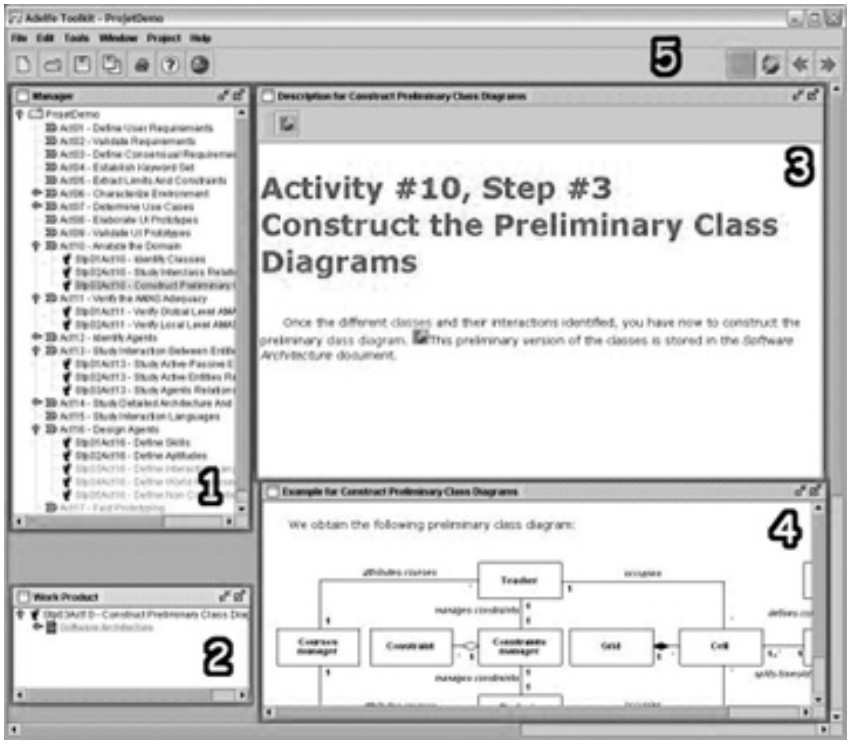
The first extension concerns the non-determinism of OR or XOR nodes. AUML remains vague concerning the decision process that manages these nodes. In ADELFE, an «aptitude»-stereotyped method is attached to the node. This method specifies the agent method that chooses the message to send or the action to do.

The second extension highlights the cooperative attitude of an agent. When receiving a message, an agent may detect a Non Cooperative Situation. To specify this detection and enable its processing, a «cooperation»-stereotyped method can be attached to the reception point of the message.

The Interactive Tool

The first functionality of the ADELFE interactive tool is to be a guide by describing the process; each stage of the process is depicted and exemplified by

Figure 2. The ADELFE Interactive Tool – A general view showing the four main windows: (1) Manager interface, (2) WorkProduct interface, (3) Description interface, and (4) Example interface. The optional Synthesis and Glossary interfaces are not shown on this figure.



Published in B. Henderson-Sellers and P. Giorgini, editors, *Agent Oriented Methodologies*, chapter 7, pages 172-202, Idea Group Publishing, 2005.

applying it to a tutorial application. This tool also provides a means to support the adopted notations and draw the needed diagrams by integrating the customized version of OpenTool. It verifies the project consistency by displaying what stages can be done depending on what has been already done or what documents have been produced so far. Finally, the AMAS adequacy tool is linked with this interactive tool to support the AMAS technology.

This interactive tool is composed of several interfaces (Bernon et al., 2004):

- A "Manager" interface (window #1, Figure 2) indicates, for the different opened projects, the different stages that designers have to follow when applying the methodology. Designers can backtrack in the methodology process as they wish, but some stages can be inaccessible (written in grey) depending on the progress state of the current opened project. Clicking on a stage name displays the related information in the other windows.
- A "WorkProduct" interface (window #2, Figure 2) dynamically lists the work products that have been produced (written in green) or that still have to be produced regarding the current progress when applying the methodology.
- A "Description" interface (window #3, Figure 2) explains stages composing the methodology process. The description text can contain flags showing that OpenTool or the AMAS adequacy tool must be used. The designer has then to click on the corresponding icon in the toolbar (#5, Figure 2) to launch it. The interactive tool is able to communicate with OpenTool to make the designer access the right diagram depending on the stage it is following.
- An "Example" interface (window #4, Figure 2) shows how the current stage has been applied to the tutorial application.
- An optional "Synthesis" interface shows a global view and an abstract of the already made stages.
- An optional "Glossary" interface explains the terms used in the methodology and defines the stereotypes that have been added to UML.

This interactive tool can be downloaded from the ADELFE Web site (*http://www.irit.fr/ADELFE*). This site constitutes an online and simplified version of the interactive tool.

Published in B. Henderson-Sellers and P. Giorgini, editors, *Agent Oriented Methodologies*, chapter 7, pages 172-202, Idea Group Publishing, 2005.

The Process of ADELFE

The primary objective of the ADELFE method is to cover all the phases of a classical software design—from the requirements to the deployment. A well-known process, the RUP, has been tailored to take into account specificities coming from the design of adaptive multi-agent systems. Phases are called WorkDefinitions (WDi), Activities (Aj) or Steps (Sk), following the vocabulary of the Object Management Group's (OMG) Software Process Engineering Metamodel (SPEM) (OMG, 2002), which has been used to express the ADELFE process (Gleizes, Millan, & Picard, 2003). Only the requirements, analysis, and design work definitions require modifications in order to be adapted to AMAS, others appearing in the RUP remaining the same. This section gives a theoretical and sequential description of these three WDs, but, of course, a designer may back track between the different stages, like in the RUP.

The stages that are specific to the AMAS technology are marked with a bold font in the description tables below. For reasons of clarity, their theoretical description is sometimes followed by a practical application to the information system case study (see first section).

WD1 & WD2 – Preliminary and Final Requirements

With respect to an object-oriented methodology, ADELFE adds nothing to preliminary requirements (WD1) as described by the RUP. The aim still consists of studying the customer needs to produce a document on which both the customer and the designer agree.

Table 1. WD1 & WD2 – Preliminary and Final Requirements in ADELFE – Their aim is to define the system such as the customer wants it to be.

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

Published in B. Henderson-Sellers and P. Giorgini, editors, *Agent Oriented Methodologies*, chapter 7, pages 172-202, Idea Group Publishing, 2005.

A6 – Characterize the Environment

Unlike classical approaches, the environment of the system is central in the AMAS theory. Actually, the adaptation process of the system depends on the interactions between the system and its environment. Therefore, during the final requirements (WD2), before determining use cases, the environment must be studied by the designer. One activity (A6) is then added to the RUP to characterize the environment of the system. This characterization begins by identifying the entities that interact with the system and constraints on these interactions (A6-S1). An entity is an actor in the UML sense and may be described as being active or passive in ADELFE. An active entity may behave autonomously and is able to act in a dynamical way with the system. A passive entity can be considered as a resource by the system; it may be used or modified by active ones but cannot change in an autonomous way. This distinction between entities is essential because agents composing the system, which are not a priori known at this stage, will be found among active ones.

In the case study, we can only find active entities, each one representing an end-user or a service provider who has subscribed to the system. An end-user seeks a relevant service provider according to his/her centres of interest, while a provider tries to find potentially interested end-users according to his/her proposed services. Due to the space limitation, the case study cannot be entirely and precisely studied in this chapter; as these two actions are totally symmetric, we will only focus on the search for a service.

In the next step (A6-S2), the context is studied through the interactions between entities and the system. This step adds no special notation and uses UML collaboration or sequence diagrams.

Finally, the designer must describe the environment with terms inspired from Russel and Norvig (1995) (A6-S3). Thus the environment may be:

- Accessible (as opposed to “inaccessible”) if the system can obtain complete, accurate, and up-to-date information about the state of its environment. For example, an environment such as the Internet is not an accessible one because knowing all about it is impossible.
- Continuous (as opposed to “discrete”) if the number of possible actions and perceptions in the environment is infinite. For example, in a real environment like the Internet, the number of actions that can be performed by users can be unbounded.
- Deterministic (as opposed to “non deterministic”) if an action has a single and certain effect. The next state of the environment is completely determined by the current state. By its very nature, the real physical world is a non-deterministic environment.

Published in B. Henderson-Sellers and P. Giorgini, editors, *Agent Oriented Methodologies*, chapter 7, pages 172-202, Idea Group Publishing, 2005.

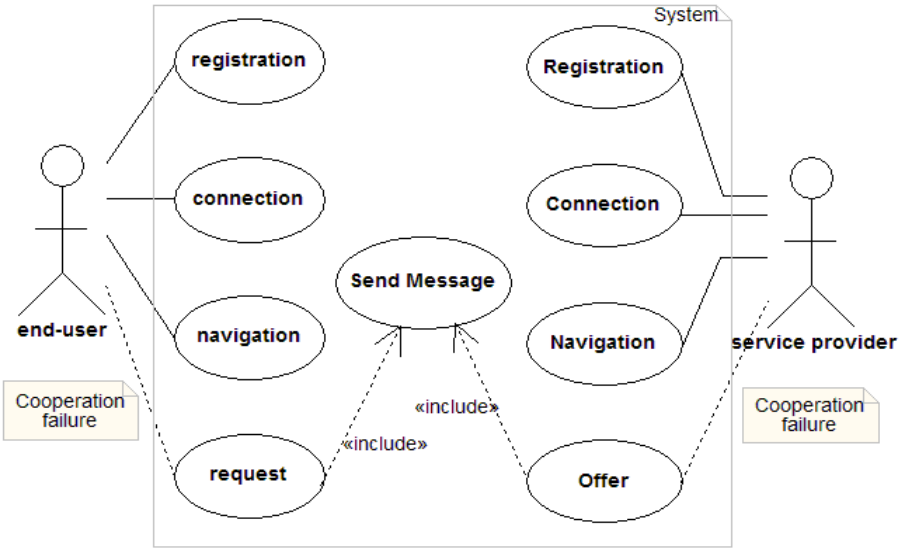
- Dynamic (as opposed to "static") if its state depends upon actions of the system that is within this environment but is also dependent on the actions of some other processes. So, changes cannot be predicted by the system. For example, the Internet is a highly dynamic environment.

In the case study, the environment of the system consists of end-users and service providers who have subscribed to the system. Each of them exerts pressure on the system (by submitting requests to find relevant service providers or to seek potential customers), and the system has to adapt itself to these constraints. Reorganizing interaction links between entities representing end-users and service providers is a way for the system to adapt to its environment. For the reasons previously given, the environment can be described as inaccessible, continuous, non-deterministic and highly dynamic.

A7- S2 – Identify Cooperation Failures

The next activity is a classical one in which use cases are identified from the point of view of the MAS user (A7). But ADELFE is only interested in "cooperative agents" that enable building AMAS. At this point, designers must also begin to think about the events that can be "unexpected" or "harmful" for the system, because these situations can lead to Non Cooperative Situations at the agent level. These "cooperation failures" can be viewed as a kind of exception. To take this aspect into account, the determination of use cases has been modified by

Figure 3. (Simplified) Use case for the information system – Two cooperation failures may be identified



Published in B. Henderson-Sellers and P. Giorgini, editors, *Agent Oriented Methodologies*, chapter 7, pages 172-202, Idea Group Publishing, 2005.

adding a step (A7-S2) in which cooperation failures must be highlighted within the previously identified use cases (A7-S1), using a specific notation (dotted arrows) added to and then supported by OpenTool.

In the case study, two main cooperation failures may occur (see Figure 3). The first one may appear during a request when the service provider replying to the end-user request is unavailable. The second may appear during an offer process if the targeted end-user has disappeared. The remainder of this second work definition is classical and will not be described here.

WD3 – Analysis

Domain analysis is a static view and an abstraction of the real world established from previous requirements and remains the same as in the RUP (A10). By studying the previously defined use cases and scenarios, the analyst identifies the components of his/her system. This identification is more or less complex depending on the studied applications (Georgé et al., 2003; Shehory & Sturm, 2001) and aims at clustering the different identified components into a preliminary class diagram.

A11 – Verify the AMAS Adequacy

As outlined in the introduction section, not every designer needs AMAS theory to build a system. Thus, a new and appropriate activity is added to the process to study the adequacy of the AMAS technology (A11) through the tool previously described. This adequacy must be studied at two levels (A11-S1 & A11-S2), through a certain number of criteria:

Table 2: WD3 – Analysis in ADELFE – Its aim is to enable the designer to structure his/her system in terms of components and interactions between these components.

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

Published in B. Henderson-Sellers and P. Giorgini, editors, *Agent Oriented Methodologies*, chapter 7, pages 172-202, Idea Group Publishing, 2005.

- At the global level, to answer the question "is a system implementation using AMAS needed?"
- At the local level, to try to answer the question "do some components need to be implemented as AMAS?" That is, is some decomposition or recursion useful during design?

If a positive answer is given by the tool in the former case, the designer can continue applying the process. Furthermore, if the tool shows the same need at the components level, ADELFE must be applied again to these components that will be considered as AMAS themselves because they need to evolve.

Even though some features of the case study application (a priori unknown algorithm, open and dynamic environment, evolutionary system, etc.) may meet the essential characteristics within the remit of the AMAS theory, the adequacy tool has been used to reinforce this idea. The values given to the different questions (see the description of the tool in the introductory section) are respectively 17, 16, 11, 19, 20, 12, 5, 19, 17, 15, and 13. Values given to criteria 1, 3, 5, and 7 express the fact that this problem is a complex and open one for which no well-established algorithm exists. The second and fourth values show that, at this point, the designer does not know exactly how to implement a solution.

The positive result given by ADELFE can be seen on the Figure 1. Using an AMAS is relevant for solving this case study; using AMAS for some components of the system is also relevant. This latter result will lead us to apply the process again once the agents have been identified (WD4) to possibly find other entities and consequently agents. We will only focus here on the entities representing end-users and service providers; other entities present in the real developed system will not be taken into account.

A12 – Identify Agents

In ADELFE, agents are not considered as being known in advance; therefore, the designer must identify them in a new activity (A12) in which the previously identified entities will be studied and evaluated. If an entity shows some specific properties (autonomy, local goal to pursue, interactions with others, partial view of its environment, ability to negotiate), it may be a potential cooperative entity (A12-S1). Indeed, this does not concern all active entities. Some of them could autonomously evolve without having a goal, for example, an autonomous resource such as pheromones or active objects. In that case, they are not potentially cooperative and will remain simple objects without becoming agents. To actually be a cooperative agent, a potential cooperative entity must be prone

Published in B. Henderson-Sellers and P. Giorgini, editors, *Agent Oriented Methodologies*, chapter 7, pages 172-202, Idea Group Publishing, 2005.

to cooperation failures. By studying its interactions with its environment and with other entities, the designer has then to determine if this entity may encounter such situations that will be considered as Non Cooperative Situations at the agent level (A12-S2). The entities verifying all these criteria will be identified as agents and the classes related to them marked with the specific «cooperative agent» stereotype (A12-S3).

In the case study, the active entities identified in A6-1 can be considered as agents because of their properties. Such entities are autonomous, have a local goal to pursue (to find a relevant service provider or to make targeted advertisement), have a partial view of their environment (other active entities), or may interact with others to target the search more effectively. They are then potentially cooperative. Furthermore, since the system is open, new entities may appear or disappear, and they may not be able to communicate as they should (e.g., an entity does not understand requests from a new one). In that case, an active entity is prone to cooperation failures and can be viewed as a cooperative agent. Each end-user (or service provider) is then represented within the system by an agent called TransactionAgent (TA).

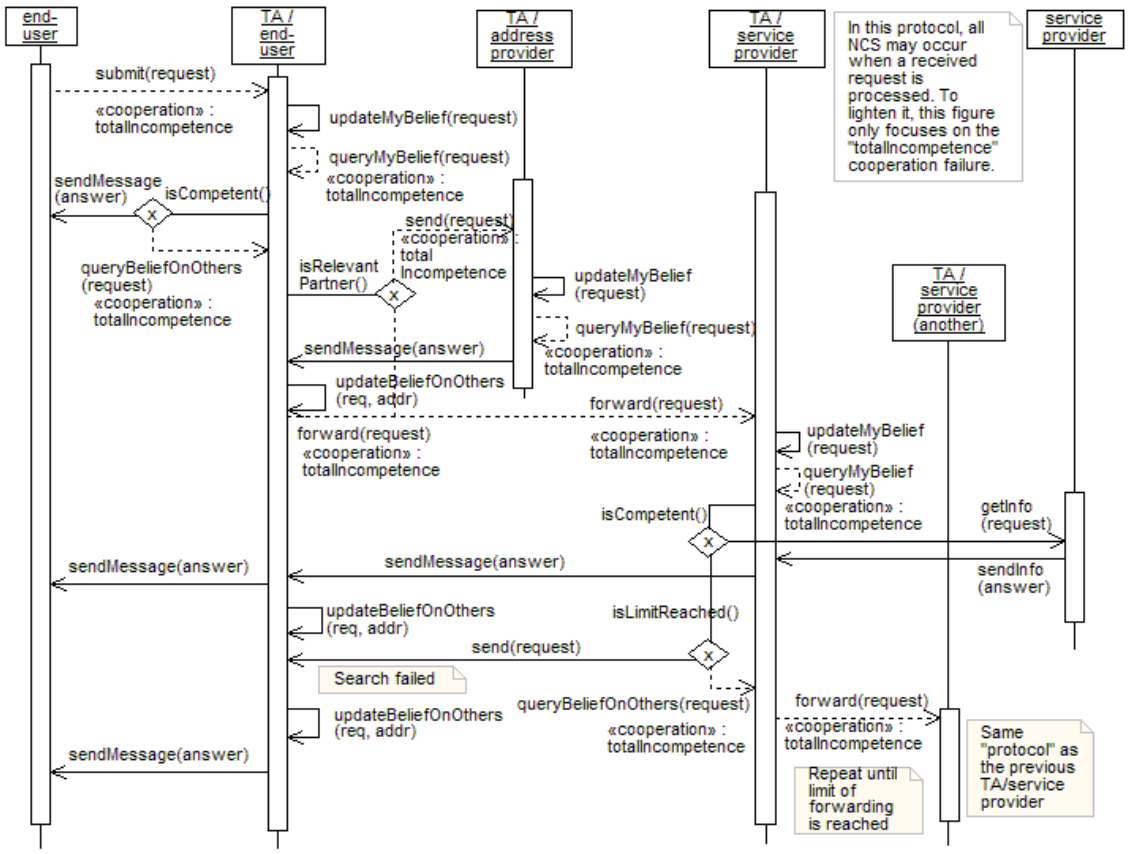
A13 - S3 – Study Agent Relationships

Interactions between all the identified entities are then studied in the last activity of this work definition (A13). Studying relationships between passive and active entities or between solely active ones is done by using UML collaboration or sequence diagrams in a standard way. However, in the last step of this activity, protocol diagrams are used to express relations between all the existing agents (A13-S3). These diagrams can be built using OpenTool modified to support AUML.

The AUML protocol diagram, shown in Figure 4, expresses the way in which the system responds to an end-user's request. The end-user TA, which represents the end-user, tries to satisfy the request by finding a relevant service provider TA. If it does not have any relevant partner, it asks a special address provider TA for partner addresses. Since every TA is cooperative, when a TA judges itself incompetent to respond to a received request, it will not discard this request; rather, it will send it again to another TA that is competent from its point of view. Every TA having the same cooperative behaviour, the request may be propagated a limited number of times, step-by-step.

Published in B. Henderson-Sellers and P. Giorgini, editors, *Agent Oriented Methodologies*, chapter 7, pages 172-202, Idea Group Publishing, 2005.

Figure 4. An AUML Protocol Diagram for the Case Study – It expresses the way in which the system answers an end-user's request.



WD4 – Design

This design work definition is an important stage because the recursive characterization of an AMAS is identified at this stage and can lead to the identification of other agents at a lower level. This may imply one design process for each new identified level of the system.

The first activity identifies the detailed architecture for the system with packages, sub-systems, objects, agents, and relations between them to produce a class diagram in which the predefined CooperativeAgent class and the «cooperative agent» stereotype will appear (A14).

In the case study, the TransactionAgent's (TA's) goal is to find relevant TAs according to the request that its associated end-user or service provider submitted for solution. To represent these TAs, a class called TA has been

Published in B. Henderson-Sellers and P. Giorgini, editors, *Agent Oriented Methodologies*, chapter 7, pages 172-202, Idea Group Publishing, 2005.

Table 3. WD4 – Design in ADELFE – Its aim is to define the system architecture.

A14: Study detailed architecture and multi-agent model S1: Determine packages S2: Determine classes S3: Use design-patterns S4: Elaborate component and class diagrams **A15: Study interaction languages**	**A16: Design agents** S1: Define skills S2: Define aptitudes S3: Define interaction languages S4: Define world representations S5: Define Non Cooperative Situations **A17: Fast prototyping** A18: Complete design diagrams S1: Enhance design diagrams S2: Design dynamic behaviours

defined. This class inherits from the CooperativeAgent class and therefore contains four mandatory methods: run, perceive, decide, and act. It is also tagged with the «cooperative agent» stereotype.

A15 – Study Interaction Languages

The designer has then to study interaction languages to define, in a new activity (A15), the way in which agents interact. If agents interact in order to communicate, for each scenario defined in A7 & A13, information exchanges between agents must be described using AUML protocol diagrams. Languages that enable interactions between agents may be implemented by a set of classes or by a design pattern, including specific agent communication tools such as an implementation of Foundation for Intelligent Physical Agents (FIPA) Agent Communication Language (ACL). As they are generic models, protocol diagrams are attached to packages and not to classes. If no direct communication exists between agents (e.g., they communicate in an indirect manner via the environment by modifying it), defining an interaction language is useless. Indeed, this step only aims at detailing the protocol used by agents to communicate and does not give any means to implement these interactions.

A16 – Design Agents

The next activity is also specific to ADELFE and added to the RUP (A16) to let the designer refine the «cooperative agent»-stereotyped classes he/she has previously defined (during A12 & A15). The different modules composing an agent must be given in this activity, as well as their physical properties (e.g.,

Published in B. Henderson-Sellers and P. Giorgini, editors, *Agent Oriented Methodologies*, chapter 7, pages 172-202, Idea Group Publishing, 2005.

weight, color, etc.). Each step composing this activity must be applied to every previously identified agent.

A16 – S1: Design Skills

Methods and attributes describing the skills of an agent must be given and stereotyped with «skill» (A16-S1).

In the case study, the skills of a TA are those of the entity it represents. For example, TAs communicate by forwarding the current request according to their representations, in order to find a relevant service provider to solve this request. When the TA of a relevant service provider is found, this TA must then use a method to ask the service provider to give a response to the request. This method getInfo is tagged with the «skill» stereotype, that is, «skill» Info getInfo (Request request).

A16 – S2: Design Aptitudes

Aptitudes of an agent must be provided, also using attributes and methods stereotyped with «aptitude» (A16-S2).

In the case study, aptitudes enable a TA to modify its representations and to interpret a received request. For example, when an end-user makes a request, his/her TA has to update its representations to learn the new centres of interest of its end-user. The method updateMyBelief enables the representations to be changed and is tagged with the «aptitude» stereotype, that is, «aptitude» int updateMyBelief (Request request).

A16 – S3: Design Interaction Languages

Among the protocols identified during A15, the designer chooses those used by an agent to interact with others. Assigning an interaction protocol to an agent automatically associates a state-machine with this agent. Attributes and methods linked with an interaction protocol must be stereotyped with «interaction» (A16-S3).

In the case study, messages exchanged between TAs deal with the requests to be resolved. Physical exchanges of these requests can be made using the mailbox concept, a buffer enabling asynchronous communication. Therefore, the attribute mailbox of a TA is tagged with the «interaction» stereotype, that is, «interaction» MailBox myMailBox.

The only way to interact is by means of message passing. The methods relating to these message exchanges that are used during the perception phase (respectively the action phase) are stereotyped with «perception» (respectively with

Published in B. Henderson-Sellers and P. Giorgini, editors, *Agent Oriented Methodologies*, chapter 7, pages 172-202, Idea Group Publishing, 2005.

«action»). For example, the method used by a TA to get a message from its mailbox is tagged by the «perception» stereotype, that is, «perception» Message getMessage().

The method to send messages is stereotyped with «action», that is, «action»void sendMessage (Message AMessage, Reference Dest).

A16 – S4: Design Representations

Attributes and methods that enable an agent to create its own representation about itself, other agents, or the environment it perceives are identified and stereotyped with «representation» (A16-S4).

In the case study, representations that an agent possesses about itself or about other TAs may evolve at runtime and they have then to be adjusted. We choose to use an AMAS to implement them. When a TA receives a request, it has to query its representation on itself to know if it is relevant to solve this request. The class TA needs the following attribute to access this component, that is, «representation» LocalBelief MyBelief.

A16 – S5: Design Characteristics

In the next step (A16-S5), the intrinsic or physical properties of an agent have to be described and tagged by the «characteristic» stereotype.

In the case study, the physical address of a TA, called myReference, represents the address of the TA in the system, that is «characteristic» Reference MyReference; and the method to get its value is: «characteristic» Reference getReference(char *Name).

A16 – S6: Design Non Cooperative Situations

This is the most important step in this activity because the ability to detect and remove Non Cooperative Situations is specific to cooperative agents (A16-S5). A model (cf. Table 4) is available to help the designer to enumerate all the situations that seem to be "harmful" for the cooperative social attitude of an agent. It reminds the designer that these situations belong to several types (such as ambiguity, uselessness, etc.) and are dependent on some conditions (one or several) that may be fulfilled or not when the agent is performing a certain phase in its lifecycle (perception, decision, action).

In the case study, if we only consider the decision phase, three NCS can be identified. All of them depend on only one condition as shown in Table 5.

After having identified every NCS an agent could encounter, the designer fills up a second type of table (see Table 6) that describes each NCS. This description

Published in B. Henderson-Sellers and P. Giorgini, editors, *Agent Oriented Methodologies*, chapter 7, pages 172-202, Idea Group Publishing, 2005.

Table 4: Generic Non Cooperative Situations – Different kinds of generic NCS exist. This table helps the designer to identify the involved ones depending on the agent's lifecycle step and the fulfilled conditions.

	Condition 1 not fulfilled		**Condition 1 fulfilled**	
	Condition 2 not fulfilled	**Condition 2 fulfilled**	**Condition 2 not fulfilled**	**Condition 2 fulfilled**
Perception	Incomprehension? Ambiguity?	Incomprehension? Ambiguity?	Incomprehension? Ambiguity?	Incomprehension? Ambiguity?
Decision	Incompetence? Unproductiveness?	Incompetence? Unproductiveness?	Incompetence? Unproductiveness?	Incompetence? Unproductiveness?
Action	Concurrence? Conflict? Uselessness?	Concurrence? Conflict? Uselessness?	Concurrence? Conflict? Uselessness?	Concurrence? Conflict? Uselessness?

Table 5: Table 4 Partially Filled up for the Case Study – Only the "decision" phase is considered.

	A TA cannot extract any informative content from the received message	*A TA can extract an informative content from only one part of the received message*	*A TA can extract several informative content from the received message*
Decision	Total Incompetence	Partial incompetence	Ambiguity

may be only a textual one to be a guide to find afterwards the methods related to the detection and removal of the NCS. This table contains:

- The state of this agent when detecting this NCS,
- A textual description of the NCS,
- Conditions describing the different elements permitting local detection of the NCS, and
- The actions linked to this NCS which describe what an agent has to do to remove it.

If the designer wants to be more precise and formal, he/she may also specify what attributes and methods will be used to express the state, conditions, and actions. Rules embedded in OpenTool will verify the consistency of their stereotyping. To express a state, only the «perception», «characteristic» or «representation» stereotypes will be used. Those stereotypes used to express conditions will be «perception» or «representation», and methods and attributes related to actions must be stereotyped with «action» or «skill».

Published in B. Henderson-Sellers and P. Giorgini, editors, *Agent Oriented Methodologies*, chapter 7, pages 172-202, Idea Group Publishing, 2005.

For the case study, the main task in this step is to fill up the table describing each NCS that a TA may encounter. Four situations have been highlighted and are textually described below.

Table 6. Description of the NCS that a TA may encounter

Name	Total incompetence
State	Receipt of a request
Description	An agent faces total incompetence when it cannot associate any meaning to the message it received: this may be due to an error in transmission or if the transmitter gets a wrong belief about it.
Conditions	During the interpretation phase the agent compares the received request with its own representation (words matching) and cannot extract any informative content from the message; it has not the necessary competence.
Actions	Because the agent is cooperative, the misunderstood message is not ignored; the agent will transmit the message to an agent that seems to be relevant according to its representations on others.

Name	Partial incompetence
State	Receipt of a request
Description	An agent is faced with partial incompetence when only one part of the received message has a meaning for it.
Conditions	During the interpretation phase the agent compares the received request with its own representation (words matching) and can extract an informative content from only a part of the message.
Actions	The receiving agent sends back the partial answer associated with the understood part of the message. It sends the other part of the request to a more relevant agent.

Name	Ambiguity
State	Receipt of a request
Description	An ambiguity occurs when the content of a received request is incomplete either because the sender gets a bad description of the receiver's tasks or because the specification of the message is wrong.
Conditions	During the interpretation phase the agent compares the received request with its own representation (words matching) and can extract several informative contents from the message.
Actions	An agent is supposed to intentionally and spontaneously send understandable data to the others. Therefore, the receiver of an ambiguous message sends back all its interpretations of the received request. The initial sender is then able to choose the most pertinent one and update its representation about the receiver's skills.

Name	Concurrence
State	Receipt of a request
Description	A situation of concurrence occurs when two agents have similar skills for a given task.
Conditions	During the interpretation phase, the agent compares the received request with its own representation (words matching). If it can extract an informative content from only a part of the request, the agent compares this request with the representation it has about other agents to find rival agents. An agent A competes with an agent B, from B's point of view, if A can extract informative content from the same part of the request as B.
Actions	Redundancy is beneficial when an agent has not been able to reach its aim or to accept a task it has been asked to undertake. In these cases, it refers the problem to its rival(s).

Published in B. Henderson-Sellers and P. Giorgini, editors, *Agent Oriented Methodologies*, chapter 7, pages 172-202, Idea Group Publishing, 2005.

For each table, at least one «cooperation»-stereotyped method has to be defined. This method corresponds to the NCS detection and will be expressed using the state and the conditions (i.e. methods and attributes) that are stereotyped with «perception», «representation» or «characteristic». If several actions are possible to remove the detected NCS, another method to choose the action to be undertaken must be defined. This method is stereotyped with «cooperation». If only one action is possible, the definition of this second method is useless: this action will always be executed.

A17 – Fast Prototyping

Once the behaviour of the agents involved in the concerned AMAS is defined, the simulation functionality of OpenTool enables the designer to test them in a new activity (A17).

This functionality of OpenTool requires a dynamic model (state-chart) for each simulated entity (object or agent). The customized version of OpenTool is able to automatically transform a protocol diagram (a particular generic sequence diagram) into a state-chart. As agents' behaviours are modelled as AIP protocol diagrams, OpenTool is then able to simulate this behaviour by running a state-machine. Therefore, uselessness or inconsistency of protocols, existence of deadlocks in these protocols, or uselessness or exhaustiveness of methods can be tested, for instance. This is done by creating the simulation environment using a UML collaboration diagram in which instances of involved agents are carrying out the generic protocol, and then, implementing some methods (using the OTScript language that is the set-based action language of OpenTool) that will be tested. If the behaviour of an agent is not adequate, the designer has to work again on it to improve it.

The last activity of the design work definition consists in finalizing the detailed architecture by enriching class diagrams (A18-S1) and then developing the state-chart diagrams that are needed to design dynamic behaviours (A18-S2). The objective is to highlight the different changes of state of an entity when it is interacting with others. For «cooperative agent»-stereotyped classes that already have state-machine(s) (coming from A16-S3), the new identified states have to appear in a new state-machine. This latter will be concurrent with the first one(s).

By now, ADELFE is only able to guide the designer until this point. The next work definitions would be those that are stipulated in the RUP: implementation and test.

Published in B. Henderson-Sellers and P. Giorgini, editors, *Agent Oriented Methodologies*, chapter 7, pages 172-202, Idea Group Publishing, 2005.

Strengths and Weaknesses of ADELFE

Strengths of ADELFE

Generally, agent-based and multi-agent based software gives a solution for complex applications in which the environment is generally constrained. However, today's and tomorrow's applications are complex and open ones; they evolve in an unpredictable environment, like the Internet, and represent the next challenge for building software. To take up this challenge, it is necessary to develop new models, tools, and methodologies.

The main strength of ADELFE (and its specificity) is to provide a methodology to design Adaptive Multi-Agent Systems coupled with a theory for those systems. According to the theory (Gleizes et al., 2000), self-organization by cooperation enables an MAS to adapt itself and to realize a function that it is not directly coded within its agents. ADELFE is thus a specialized methodology that deals with only a certain kind of agents—cooperative ones—to build systems in which the global function emerges from the interactions between these agents.

ADELFE is based on "standards" such as the RUP, UML, or AUML notations to promote agent-oriented programming in an industrial world where object-oriented software engineering is the norm.

Furthermore, as previously seen, ADELFE provides some tools, notably the interactive tool that helps the designer to not only follow and apply the process but also to produce the different artifacts needed during the process lifecycle. OpenTool, the graphical modelling tool linked with ADELFE, supports the UML notation and has been modified to integrate some AUML diagrams. For industry, it is very important to know very early whether the system to be developed justifies some investment in a new methodology or technique. Therefore, ADELFE guides the developer in making the decision as to if and where AMAS technology is required in the system being developed. This explains the importance of the adequacy checking in the analysis workflow and the adequacy tool that analyses criteria given by the designer to decide if this technology is useful. If the application is not suited to AMAS technology, the designer could use another agent-oriented methodology.

ADELFE does not suppose that the agents in the designed system are known in advance and offers a specific activity and some criteria to help the designer to identify what entities in the system require implementation as agents. To decide whether entities should be considered as agents, their features must be studied as well as the interactions and the cooperation failures they may have.

Published in B. Henderson-Sellers and P. Giorgini, editors, *Agent Oriented Methodologies*, chapter 7, pages 172-202, Idea Group Publishing, 2005.

The autonomous behaviour of an agent, which results from its perceptions, knowledge, and beliefs, is very difficult to define in complex systems and in dynamic systems. Actually it is very difficult to enumerate all possible actions for each state of the environment. ADELFE specifically deals with AMAS; following the theory, it only deals with a specific kind of agent architecture. The methodology then provides cooperative agent architecture and some means to endow an agent with a cooperative behaviour. Skills or representations of a cooperative agent may evolve if this agent has to adjust them. In that case, they will be implemented using AMAS, and the developer may reuse the entire methodology to develop a part of the behaviour of an agent — ADELFE is a recursive or iterative methodology. Finally, the greatest difficulty in this behaviour definition is to identify Non Cooperative Situations that an agent may encounter, and some models are given to help the designer to find these.

An automatic transformation from collaboration diagrams into state-machines has been added to OpenTool to allow their simulation. Once the agents are defined, this specific activity may be used to test the behaviour of an agent to improve it if needed.

Modularity is also an important strength of ADELFE. It has been based on an interpretation of the RUP by adding some specific activities and steps, namely, those related to the AMAS technology. The process of ADELFE can then be decomposed into fragments that may be reused in other agent-oriented methodologies[3]. It would also be easier to integrate pieces coming from other methodologies into ADELFE. For instance, if the AMAS technology is useless, it would be interesting to guide the designer towards a step of another methodology more suited for his/her problem.

Limitations of ADELFE

The main strength of ADELFE could also be its major limitation; it is specialized and therefore cannot be used to design all the existing applications or to model all types of agents (e.g., BDI). For instance, to design a system such as simulation software, embedding it within another system such as a simulation platform, would be needed. Preliminary steps would therefore be required to design the whole simulation software by studying the most external system and expressing the needs of the user simulation (such as statistical results, observation algorithms, etc.). Nevertheless, this limitation by specialization is lessened by integrating the AMAS adequacy verification activity into the process. Furthermore, this limit could also be removed by coupling another methodology with ADELFE. It would then be interesting to have a more general methodology

Published in B. Henderson-Sellers and P. Giorgini, editors, *Agent Oriented Methodologies*, chapter 7, pages 172-202, Idea Group Publishing, 2005.

coupled with ADELFE in order to take into account both problem solving and simulation software design.

Some activities could be improved, especially the fast prototyping one. For the moment, it only enables the designer to test the behaviour of the defined agents and validate them according to the specifications. We would like to improve this activity to provide greater help during the design and implementation of agents. The designer would be able to interact with the system during its design for improving the behaviour of agents by adding or removing some parts of it.

Some work definitions are still lacking. At the present time, no operational tool such as a platform or a set of software tools is coupled to ADELFE to guide implementation, testing, or deployment.

The interactive tool is linked with OpenTool and verifies the production of artifacts to make the designer progress in the process application. Although there is no automated tool for consistency checking of the different activities results, it is a future targeted improvement.

Finally a disadvantage of ADELFE is common to all design methods, the graphical modelling tool is complex, and sometimes the designer may find it difficult to use.

Purposeful Omissions

Some purposeful omissions were made in ADELFE, mainly due to the fact that ADELFE tries to constrain the agent behaviour with a cooperative attitude.

Therefore, the role notion is useless because designers have only to focus on the ability an agent possesses to detect and solve cooperation failures. If a designer gives roles to agents, by describing a task or protocols, he/she will establish a fixed organization for these agents. However, a fixed organization in an AMAS is not welcomed because this organization must evolve to allow the system to adapt.

Neither does the goal notion appear. The goal an agent has to achieve is modelled by its skills, aptitudes, and representations; using the term “goal” in one of the ADELFE models is not useful.

Using an ontology can be motivated by the agent granularity and may become useful if agents are coarse-grained. But according to the AMAS theory, if agents have to adapt themselves to their environment, they are also able to adapt to the other agents. This adaptation can lead agents to learn to understand each other making the use of ontology not essential in ADELFE.

Published in B. Henderson-Sellers and P. Giorgini, editors, *Agent Oriented Methodologies*, chapter 7, pages 172-202, Idea Group Publishing, 2005.

Conclusion

This chapter devoted to the ADELFE methodology has first presented the theoretical background that led to the development of this specialized methodology. Unpredictable situations due to interactions between agents or to the openness of the environment are locally processed by agents composing the system. Locally processing these cooperation failures is enough to change the system organization, without relying on the knowledge of the global function that must be obtained, and therefore to make the system adapt.

The ADELFE toolkit also provides some tools that have been briefly presented in a second part: an interactive tool to help the designer to follow the process, a graphical modelling tool to support the use of the dedicated notation, and an AMAS adequacy tool to warn the designer if the problem is not suited to this kind of technology.

The process of ADELFE is based on the RUP and uses UML and AUML notations. It has been described in the third section of this chapter, using an information system case study to better visualize how to apply the different stages.

ADELFE aims at promoting a specific kind of MAS and is not a general methodology. This specificity is its main strength but also one of its limitations. Therefore, a first perspective of this work would be to define “fragments” that could be interrelated with others, coming from different complementary methodologies. That would enable a designer to build his/her own methodology (adapted to his/her particular needs) from different existing ones. A second perspective of our work in the engineering domain would also be to endow ADELFE with a tool that would automatically transform a meta-model into a model depending on a target platform in the spirit of OMG’s Model-Driven Architecture (MDA) initiative[4].

Acknowledgements

We would like to thank the support of the French Ministry of Economy, Finance, and Industry, as well as our partners,: TNI-Valiosys Ltd. (for their customization of Open-Tool), ARTAL technologies Ltd., and the IRIT software engineering team (for their help and work on UML and SPEM).

Published in B. Henderson-Sellers and P. Giorgini, editors, *Agent Oriented Methodologies*, chapter 7, pages 172-202, Idea Group Publishing, 2005.

References

Bernon, C., Camps, V., Gleizes, M-P., & Picard, G. (2004). Tools for self-organizing applications engineering. In G. Di Marzo Serugendo, A. Karageorgos, O. F. Rana, & F. Zambonelli (Eds.), *First International Workshop on Engineering Self-Organizing Applications (ESOA)*, Melbourne, Australia, LNCS 2977, Berlin: Springer-Verlag.

Bernon, C., Gleizes, M-P., Peyruqueou, S., & Picard, G. (2002). ADELFE, a methodology for adaptive multi-agent systems engineering. In P. Petta, R. Tolksdorf, & F. Zambonelli (Eds.), *Third International Workshop "Engineering Societies in the Agents World" (ESAW)*, LNAI 2577, pp. 156-169. Berlin: Springer-Verlag.

Brazier, F. M. T., Jonker, C. M., & Treur, J. (1999). Compositional design and reuse of a Generic agent model. In *Proceeding of Knowledge Acquisition Workshop (KAW'99)*.

Camps, V., Gleizes, M-P., & Glize, P., (1998). A self-organization process based on cooperation theory for adaptive artificial systems. In *Proceedings of the First International Conference on Philosophy and Computer Science "Processes of Evolution in Real and Virtual Systems"*, pp. 2-4, Krakow, Poland.

Capera, D., Georgé, J-P., Gleizes, M-P., & Glize, P. (2003). The AMAS theory for complex problem solving based on self-organizing cooperative agents. In *Proceedings of the First International Workshop on Theory and Practice of Open Computational Systems* (TAPOCS03@WETICE 2003), Linz, Austria.

Desfray, P., (2000). 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.

Georgé J-P., Gleizes, M-P., Glize, P., & Régis, C., (2003). Real-time simulation for flood forecast: An adaptive multi-agent system STAFF. In *Proceedings of the AISB'03 symposium on Adaptive Agents and Multi-Agent Systems*, Univ. of Wales, Aberystwyth.

Gleizes, M-P., Georgé, J-P., & Glize, P., (2000). A theory of complex adaptive systems based on co-operative self-organisation: Demonstration in electronic commerce. In P*roceedings of the Self-Organisation in Multi-Agent Systems Workshop (SOMAS)*, Milton Keynes, UK.

Gleizes, M-P., Glize, P. & Link-Pezet, J., (2003). An adaptive multi-agent tool for electronic commerce. In the *Proceedings of the IEEE 9th Interna-*

Published in B. Henderson-Sellers and P. Giorgini, editors, *Agent Oriented Methodologies*, chapter 7, pages 172-202, Idea Group Publishing, 2005.

tional Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE'00), Gaithersburg, Maryland.

Gleizes, M-P., Millan, T., & Picard, G., (2003a). ADELFE: Using SPEM notation to unify agent engineering processes and methodology. *IRIT Internal Report* 2003-10-R.

Iglesias, C.A., Garijo, M. & Gonzalez, J.C., (1998). A survey of agent-oriented methodologies. In *Proceedings of the Fifth International Workshop on Intelligent Agents V, Agent Theories, Architectures, and Languages*, pp. 317-330.

Kruchten, P. (2000). *The rational unified process: An introduction.* Reading, MA: Addison Wesley.

Odell, J., Van Dyke Parunak, H., & Bauer, B., (2000). Extending UML for agents. In *Proceedings of the Agent Oriented Information Systems (AOIS) Workshop at the 17th National Conference on Artificial Intelligence (AAAI).*

Odell, J., Van Dyke Parunak, H., & Bauer, B., (2001). Representing agent interaction protocols in UML. In *Proceedings of the First International Workshop on Agent-Oriented Software Engineering (AOSE'00)*, Limerick, Ireland, pp. 121-140. Berlin: Springer-Verlag.

OMG, (2002). *Software Process Engineering Metamodel Specification*. Retrieved from: *http://cgi.omg.org/docs/formal/02-11-14.pdf*

Picard, G. (2003). UML stereotypes definition and AUML notations for ADELFE methodology with OpenTool. *First European Workshop on Multi-Agent Systems (EUMAS'03)*. Oxford, UK.

Piquemal-Baluard, C., Camps, V., Gleizes, M.-P., & Glize, P., (1996). Properties of individual cooperative attitude for collective learning. In *Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'96)*, Eindhoven.

Russel, S. & Norvig, P., (1995). *Artificial intelligence: A modern approach*. Englewood Cliffs, NJ: Prentice Hall Series.

Shehory, O. & Sturm, A., (2001). Evaluation of modeling techniques for agent-based systems. In *Proceedings of the Fifth International Conference on Autonomous Agents*, pp. 624-631. New York: ACM Press.

Topin, X., Fourcassie, V., Gleizes, M-P., Theraulaz, G., Régis, C., & Glize, P., (1999). Theories and experiments on emergent behaviour: From natural to artificial systems and back. In *Proceedings of the European Conference on Cognitive Science*, Siena.

Wood, M. & DeLoach, S. (2000). An overview of the multiagent systems engineering methodology. In *Proceedings of the First International*

Published in B. Henderson-Sellers and P. Giorgini, editors, *Agent Oriented Methodologies*, chapter 7, pages 172-202, Idea Group Publishing, 2005.

Workshop on Agent-Oriented Software Engineering (AOSE'00), Limerick, Ireland, pp. 207-221. Berlin: Springer- Verlag.

Wooldridge, M. (2002). *An introduction to multi-agent systems.* New York: Wiley.

Endnotes

1 Agents and MASs notions are introduced in Chapter 1.

2 See Chapter 1 for more details about modelling languages.

3 Ed. As will be discussed in Chapter 13.

4 Some hints about MDA are given in Chapter 1.