Published in F. Bergenti, M.-P. Gleizes, and F. Zambonelli, editors, *Methodologies and Software Engineering for Agent Systems*, pages 157-176. Kluwer Publishing, 2004.

Chapter 8

THE ADELFE METHODOLOGY

Designing Adaptive Cooperative Multi-Agent Systems

Gauthier Picard

Marie-Pierre Gleizes
Institut de Recherche en Informatique de Toulouse (CNRS - INP - UPS)
118, route de Narbonne - 31062 Toulouse Cedex, France
{picard, gleizes}@irit.fr

Abstract This paper presents a method named ADELFE, which is led by the Rational Unified Process but is devoted to software engineering of adaptive multi-agent systems. ADELFE guarantees that the software is developed according to the AMAS theory[1]. We focus this presentation on the four first core workflows of the RUP. Therefore, in the preliminary requirements an agreement on what the system has to do must be reached. During the final requirements phase, the environment of the studied system must be defined and characterized. Then, in the analysis phase, the engineer is guided to decide to use adaptive multi-agent technology and to identify the agents through the system and the environment models. Finally, the design workflow of ADELFE must provide the cooperative agent's model and helps the developer to define the local agents' behavior. We illustrate the methodology by applying it to a case study: a timetable design.

Keywords: Adaptive Multi-Agent System, Self-organization, Agent-Oriented Methodology, Cooperation, Emergence.

[1]The Adaptive Multi-Agent Systems (AMAS) theory has been in development and applied for the last 8 years at the Research Institute in Computer Science of Toulouse (IRIT). See http://www.irit.fr/SMAC

Published in F. Bergenti, M.-P. Gleizes, and F. Zambonelli, editors, *Methodologies and Software Engineering for Agent Systems*, pages 157-176. Kluwer Publishing, 2004.

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 more importantly an a priori known algorithm to find a solution does not exist. This solution must build itself according to interactions the system will have with its environment during its functioning. Classical approaches to solve problems in such a context cannot be applied. That led us to propose a theory called AMAS (Adaptive Multi-Agent System) theory [5], 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, an electronic commerce tool for mediation of services, a software tool for adaptive flood forecast or 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 method 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 is a toolkit to develop software with emergent functionality [2]. ADELFE is not a general method; it concerns applications in which self-organization makes the solution emerge from the interactions of its parts. It guarantees that the software is developed according to the AMAS theory. It also gives some hints to the designer to tell him if using the AMAS theory is relevant to build his application. The ADELFE toolkit enables the development of software with emergent functionality and consists of a software development process, a notation based on UML / AUML, some tools supporting the process, and the notations and a library of components that can be used to make the application development easier.

This paper is structured as follow: section 2 gives principal concepts of the AMAS theory and a brief overview of the methodology and of the case study: ETTO (Emergent Time Tabling Organization) application used to illustrate the process. Then, sections 3, 4, 5 and 6 detail respectively the requirement, the specification and the design work definitions. Section 7 presents the main tools associated with ADELFE. Before concluding, section 8 gives a brief comparison to some other well-known methodologies.

2. ADELFE Methodology Overview

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

Published in F. Bergenti, M.-P. Gleizes, and F. Zambonelli, editors, *Methodologies and Software Engineering for Agent Systems*, pages 157-176. Kluwer Publishing, 2004.

Then, the requirements for the timetabling problem[2] case study used to illustrate the methodology are presented.

2.1 The AMAS Theory

The AMAS theory provides a solution to build complex systems for which classical algorithmic solutions cannot be applied [6, 5]. 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. 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 unpredictable situations (like conflict, concurrency, etc.), called Non Cooperative Situations (NCS). Faced with a NCS, a cooperative agent acts to reach a new 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 behavior.

Applying the AMAS theory consists in enumerating, according to the current problem to solve, all the cooperative 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 Overview

The ADELFE process consists in six work definitions: Preliminary Requirements, Final Requirements, Analysis, Design, Implementation and Tests.

Among the different activities or steps that are listed in the figure 8.1, some are marked with a bold font to show that they are specific to adaptive multi-agent systems. Only the four work definitions require modifications in order to be tailored to AMAS and the main activities are presented in the next sections. Other work definitions appearing in the RUP remain the same [11].

[2]This example was elaborated as a case study to compare and discuss different methodologies and multi-agent platforms for the ASA Group of the French Artificial Intelligence Association.

Published in F. Bergenti, M.-P. Gleizes, and F. Zambonelli, editors, *Methodologies and Software Engineering for Agent Systems*, pages 157-176. Kluwer Publishing, 2004.

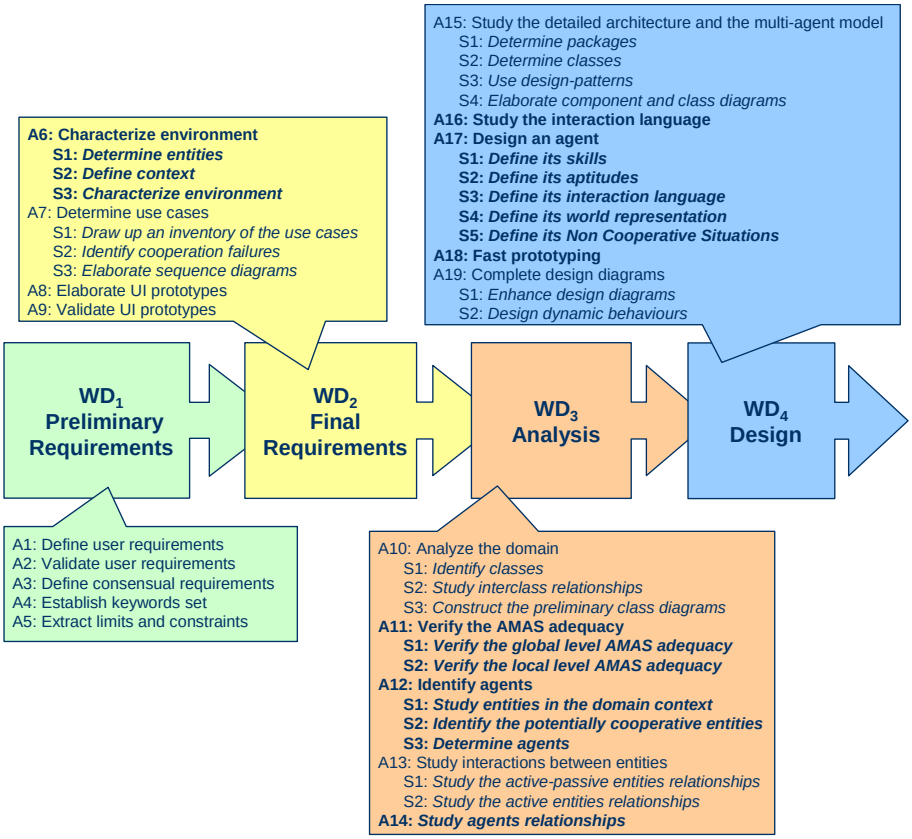

Figure 8.1. ADELFE process is described in three levels: *Work Definitions* (WD_i), *Activities* (A_j) and *Steps* (S_k).

2.3 The ETTO Case Study

In order to show how a self-organizing application can be developed using the tools linked with ADELFE, the next sections will refer to the ETTO (or Emergent Time Tabling Organization) application. Description and design of the system related to ETTO are not the main objective of this article and more information is available in [2]. The chosen problem is a classical course timetabling one in which time slots and locations (rooms) must be assigned to teachers and students groups in order to let them meet during lectures. Usually, solutions to such a problem can be found using different techniques like constraint-based ones or meta-heuristics techniques (simulated annealing, taboo search, graph coloring, etc.) and more recently, neural networks, evolutionary or ant algorithms. However, no real solving technique exists when the constraints can dynamically evolve and when the system needs to adapt. Be-

Published in F. Bergenti, M.-P. Gleizes, and F. Zambonelli, editors, *Methodologies and Software Engineering for Agent Systems*, pages 157-176. Kluwer Publishing, 2004.

cause this problem is not a simulation one, because it is actually complex when handmade or not (it belongs to the NP-complete class of problems) and has no universal solution, we do think that it represents the right example to apply the AMAS theory. The aim is to make a solution emerge, at the macro-level of the built MAS, from the interactions of independent parts at the micro-level.

General requirements for this ETTO problem are the following. Stakeholders are teachers, students groups and lecture rooms. Every actor individually owns some constraints that must be (at best) fulfilled. A teacher has some constraints about his availabilities (e.g. the days or the time slots during which he can teach), his capabilities (e.g. the topics he can lecture on) and the needs he has about particular pedagogic equipments (overhead projectors, video projectors, a defined lecture room for a practical work, etc.). A students group must take a particular teaching made up of a certain number of time slots for a certain number of teaching topics (X time slots for a topic 1, Y time slots for a topic 2, etc.). A lecture room is equipped or not with specific equipments (an overhead projector, a video projector or any equipment for practical works) and can be occupied or not (e.g. during a given time slot or on a certain day).

3. Preliminary Requirements

The aim of the preliminary requirements is to define the system to be and to establish an agreement on the preliminary requirements.

The preliminary requirements work definition concerns the description of the system and the environment in which the system will be deployed. It consists in defining what to build or what is the most appropriate system for end-users. End-users, clients, analysts and designers have to list the potential requirements, to define the context in which the system will be deployed and to list the functional and non-functional requirements (Activity #1 and #3). They must agree on these requirements (Activity #2 and #3). Then, designers have to define the main concepts used to describe the application and its domain (the system and its environment) (Activity #4). And they must define the limits and constraints of the system they have to build (Activity #5).

4. Final Requirements

The aim of the final requirements is to transform this view in a use-case model, and to organize and to manage the requirements (functional or not) and their priorities. In fact, at this stage, the designer has to define the function of the studied system and to model its environment. To take into account adaptive multi-agent systems, four steps are added to the RUP process: three are in the Activity #6 (Characterize environment) and one is in the Activity #7 (Determine Use cases). The two last activities in this workflow relating to User Interface elaboration are not described here.

Published in F. Bergenti, M.-P. Gleizes, and F. Zambonelli, editors, *Methodologies and Software Engineering for Agent Systems*, pages 157-176. Kluwer Publishing, 2004.

4.1 Characterization of the Environment

This activity (Activity #6 in figure 8.1) is divided into three steps: determining the entities that are involved in the system, defining the context and characterizing the environment. Entities to identify are active or passive entities in interaction with the system.

A detailed definition of the system environment is necessary to develop adaptive systems, which are able to respond to any change. This step firstly focuses on what may be in interaction with the studied system in terms of passive or active entities, or constraints. In our example, teachers, students, the planning manager and the room manager are active entities because they are able to change by themselves their own constraints or they can interact with the system. Rooms are passive because they represent resources and they cannot modify their characteristics by themselves. The NPP (or National Pedagogic Plan) is the database that contains all information concerning the courses (maximum number of sessions per week, hour quotas for each formation, etc): it is a passive entity.

In a second time, this step must define the context of the system. It requires a characterization of data streams and interactions between entities and the system. Data streams between passive entities and the system are expressed using collaboration diagrams. Interactions between active entities and the system are expressed using sequence diagrams.

In our example, two kinds of data flows between the system and passive entities exist: when the system consults the NPP and when the system consults room constraints. When an active entity wants to interact with the system, it may only have to change constraints (owner constraints or room constraints). In the other sense, the system interacts with the active entity by displaying the planning.

Finally, to characterize the environment, designers must think about the environment of the system to build in terms of being accessible or not, deterministic or not, dynamic or static and discrete or continuous. These terms have been reused from [14] and represent a help to later determine if the AMAS technology is needed or not. This characterization may enable the designer to detect some special use cases to respond to environment behavior. In the case study, the environment of the system can be characterized as follow:

- *Dynamic*: the evolution of the active entities does not depend on the system, they are unpredictable from the point of view of the system;
- *Accessible*: the environment can obtain information on the state of the environment;
- *Non-deterministic*: the system is not able to know what could be the effects of its actions on active entities;

Published in F. Bergenti, M.-P. Gleizes, and F. Zambonelli, editors, *Methodologies and Software Engineering for Agent Systems*, pages 157-176. Kluwer Publishing, 2004.

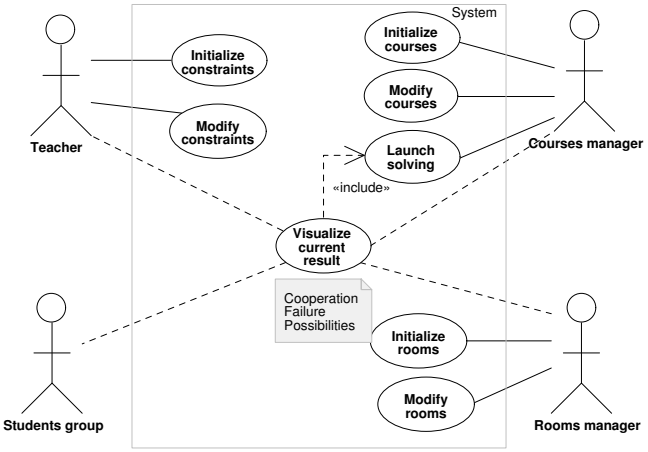

Figure 8.2. The use cases for the ETTO problem. Associations to Visualize current result case are potentially non cooperative: the result of the time tabling resolution is the only cause of cooperation failure between the users and the system, in the sense that users expect the system to satisfy their constraints.

- *Continuous*: the number of interactions between system and entities are infinite.

4.2 Determination of the Use Cases

The main objective of this activity, which ends the requirements workflow, is to clarify the different functionalities the system has to respond to. This activity is divided into three steps which enable to design use cases, to elaborate the associated sequence diagrams and to identify cooperation failures. Only active entities are implied in these use cases, which are the results of a functional requirements set. Identification of cooperation failures between the system and its environment is realized in order to help designers to detect problems in the sense of the AMAS theory : Non Cooperative Situations. This identification will be refined during the development process and enables identification of agents later in the process. The use cases for the timetabling problem are shown in figure 8.2. Cooperation failures are represented on use cases diagrams by dotted lines.

5. Analysis

From a multi-agent point of view, the identification of the agents must take place in this workflow. The analysis work definition has to develop an understanding of the system, its structure in terms of components and to know if the AMAS theory is required.

Published in F. Bergenti, M.-P. Gleizes, and F. Zambonelli, editors, *Methodologies and Software Engineering for Agent Systems*, pages 157-176. Kluwer Publishing, 2004.

5.1 Domain Analysis

Domain analysis (Activity #10) is a static view and an abstraction of the real world and the linked entities. Considering separately each use-case by defining scenarios, the designer has to divide the system into entities. The result of this step is a set of entities in preliminary class diagrams. Teacher, CourseManager, StudentGroup, Room, RoomManager and NPP classes appear naturally as real world entities. In a second time, we tried to determine what entities could be useful for our system. A board is proposed to visualize the organization (Grid and Cell classes) and the ConstraintManager class to control constraints for each entity that owns a Constraint class instance. Cells represent intersections of different dimensions (days, rooms, etc).

5.2 Adequacy of the AMAS Theory

This activity (Activity #10 in figure 8.1) aims to help the designer to decide if the AMAS theory is adequate to solve his problem because, for certain applications, this kind of programming can be useless. A software component has been developed with several criteria to study the adequacy at two levels :

- 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?” i.e. is some decomposition or recursion useful during design?

For the case study, the decision tool clearly suggests to use the AMAS to design the global level. Moreover, the tool indicates that some entities could be decomposed as AMAS. So, once the agents are identified, the designer has to reuse the method on them, as developed below.

5.3 Agent Identification

In this activity (Activity #12), we are only interested in agents that enable a designer to build our sort of AMAS. The designer has to determine which entities fit with this agent type: “cooperative agents”. A cooperative agent ignores the global function of the system; it only pursues an individual objective and tries to be permanently cooperative with other agents involved in the system. The global function of the system is emerging (from the agent level to the multi-agent level) thanks to these cooperative interactions between agents. An agent can also detect Non Cooperative Situations (NCS) that are situations it judges as being harmful for the interactions it possesses with others, such as lack of understanding, ambiguity or uselessness. This does not mean that an agent is altruistic or always helping other agents but it is just trying to have use-

Published in F. Bergenti, M.-P. Gleizes, and F. Zambonelli, editors, *Methodologies and Software Engineering for Agent Systems*, pages 157-176. Kluwer Publishing, 2004.

ful (from its point of view) interactions with others. Thus, facing up to a NCS, an agent always acts to come back to a cooperative state. In fact, the behavior of the collective compels the behavior of an agent. In ADELFE, designers are given some guidelines: an entity can become an agent if it can be faced with "cooperation failures" and may have evolutionary representations about itself, other entities or about its environment and/or may have evolutionary skills. At this stage, we identify teachers and students groups as being cooperative agents. All other entities are considered as objects.

5.4 Adequacy of the AMAS Theory at the Local Level

If the first step of adequacy to the AMAS theory indicates a possible decomposition, each agent has to be analyzed as a system. The goals of an agent, Teacher or StudentGroup, are to find different places and partners to follow or to give each course. These goals raise the problem of ubiquity. Agents cannot be at different places at different moments. Therefore, we propose to create one agent per course for each teacher or student group. Two agent levels are distinguished:

- RepresentativeAgent (RA): at the highest level, it represents a teacher or a student group within the system;

- BookingAgent (BA): at the lowest level, it is responsible for finding partners and booking rooms for a RA. There are as many BA as the number of courses a teacher has to give or a student group has to follow.

The identified agents have to be added to the preliminary class diagram as shown in figure 8.3.

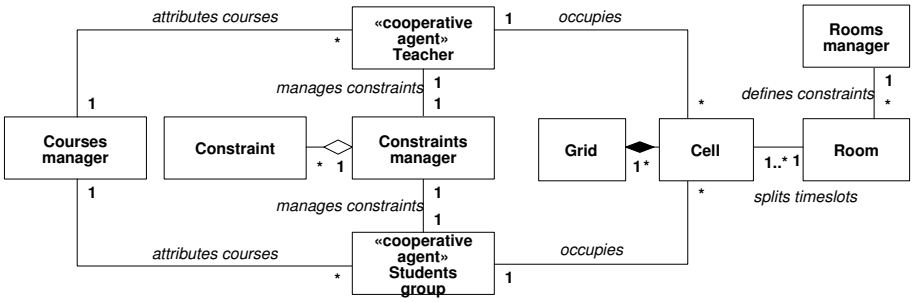

Figure 8.3. The main class diagram for the ETTO problem. Three classes of agents appear: the StudentsGroup, the Teacher and the BookingAgent. The two firsts are interface agents between the system and the users. BookingAgents aim to reserve time slots (Cell of a Grid) and Rooms for Teachers or StudentsGroups in terms of their Constraints.

Published in F. Bergenti, M.-P. Gleizes, and F. Zambonelli, editors, *Methodologies and Software Engineering for Agent Systems*, pages 157-176. Kluwer Publishing, 2004.

x

6. Design

The design work definition aims to formulate models that focus on non functional requirements and the solution domain and that prepare for the implementation and test of the system. In ADELFE, agents being identified and their relationships being studied, designers have now to study the way in which the agents are going to interact (Activity #15) thanks to protocol diagrams. ADELFE also provides a model for designing cooperative agents (Activity #16); designers must describe, for each type of agents, its skills, its aptitudes, its interaction language, its world representation and the Non Cooperative Situations this agent can encounter. The global function of a self-organizing system is not coded; designers have only to code the local behavior of the parts composing it. An activity of fast prototyping (Activity #17) based on finite state machines has been added to the process. It enables designers to verify the behavior of the agents being built. Then designers have to complete the previous defined class diagram (Activity #18). Once a class diagram enhanced, finalized indeed, this diagram may require the development of a statechart diagram. The aim is to highlight the different state changes of an entity when it is interacting with others.

Because the complete design cannot be described in this paper, we only detail the agent design activity which does not exist in other methodologies. Five steps compose this activity, the fourth one enables to endow an agent with classical parts such as: skills, aptitudes, an interaction language and world representations and the last one is more specific to AMAS theory.

6.1 Study of Interaction Languages

The result of this activity (Activity #15) is a set of protocol diagrams representing the different interaction languages that may be used by the agents. Figure 8.4 shows a sample protocol between two BookingAgents having two different roles: ExploringTeacherAgent and OccupyingTeacherAgent. The AUML – Agent Unified Modeling Language [13] – notation is used but some specific functionality has been added to AIP diagrams to fit to the AMAS theory requirements. The decision-making process corresponding to an OR or a XOR branch is done by an «aptitude»-stereotyped operation attached to the branch-node (see §6.2). For example, in figure 8.4, the isRoomFitting operation is attached to the first XOR node ; i.e. depending on the results of this operation, the ExploringTeacherAgent may either request for more information before resuming its exploration of the grid, or negotiate in terms of the constraints which agents own.

Once the set of protocols defined, designers may assign them to agents during the new activity, as these are generic. Another possibility is to specify fully generic protocols in which only roles are manipulated.

Published in F. Bergenti, M.-P. Gleizes, and F. Zambonelli, editors, *Methodologies and Software Engineering for Agent Systems*, pages 157-176. Kluwer Publishing, 2004.

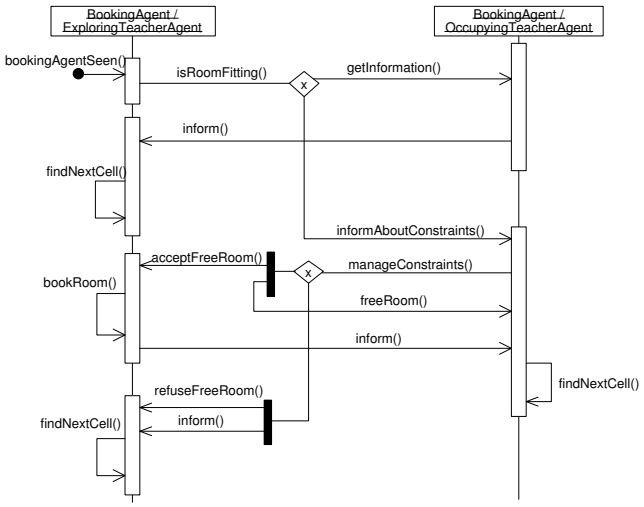

Figure 8.4. An example of protocol diagram between two BookingAgents of two different Teachers. The first agent explores the time-table grid to find satisfying slots and rooms. The second one already occupies a room and a slot. This diagram explains the negotiation between these two agents when the first agent meets the second agent. This negotiation may either result on the leaving of the first agent or the booking, by the first agent, and the leaving of the second agent.

6.2 Agent Design

This activity helps the designer to fill in a generic architecture given for an agent used in the AMAS theory. ADELFE is a method which is devoted to a specific kind of agents: cooperative ones. Therefore, even if an agent still follows the same defined life cycle – 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 – 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.

Published in F. Bergenti, M.-P. Gleizes, and F. Zambonelli, editors, *Methodologies and Software Engineering for Agent Systems*, pages 157-176. Kluwer Publishing, 2004.

- *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.

Adelfe Stereotypes. To enable the developer to deal with these specific components in the ADELFE methodology, nine stereotypes have been defined to express how an agent is formed and/or how its behavior 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 and their rules (written in OTScript language) are included in the OpenTool graphical development tool linked with ADELFE. All these stereotypes, except «cooperative agent», can be applied to attributes and/or methods. All the examples appearing in this section refer to the figure 8.5.

The «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 im-

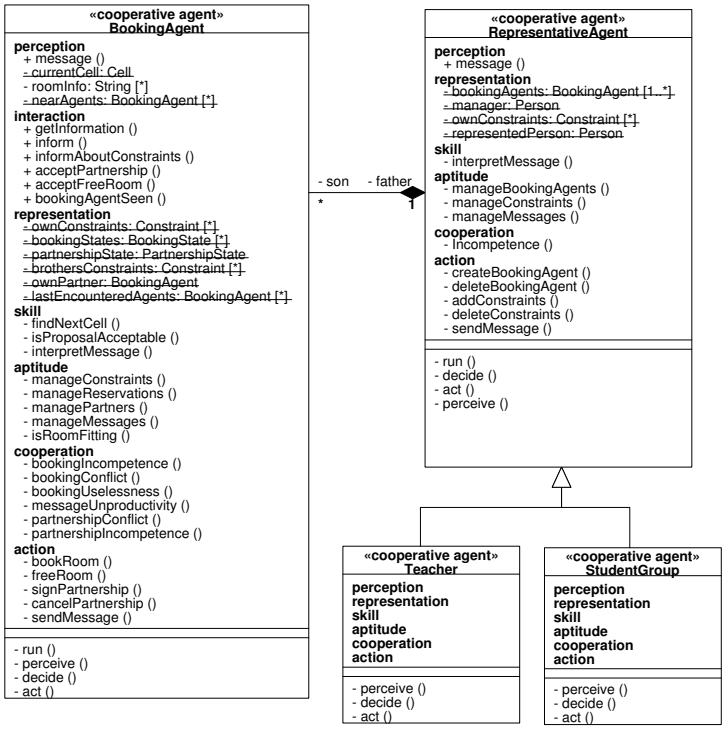

Figure 8.5. The two main «cooperative agent»-stereotyped classes : RepresentativeAgent and BookingAgent. The first one can represent either a StudentGroup or a Teacher.

Published in F. Bergenti, M.-P. Gleizes, and F. Zambonelli, editors, *Methodologies and Software Engineering for Agent Systems*, pages 157-176. Kluwer Publishing, 2004.

plemented using a class that will be stereotyped with «cooperative agent». This class must have a run method that simulates the agent's life cycle. Therefore, to ensure that this method does exist, an agent inherits from a superclass called CooperativeAgent. A sample associated coherency rule is: an agent-stereotyped class inherits (directly or not) from the CooperativeAgent class. For example, 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 time slots in the timetable. A BookingAgent class 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.

The «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 life cycle. It can also be accessed or called by other agents.

The «perception» stereotype expresses how an agent 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. A associated coherency rule is: "an attribute stereotyped with «perception» is necessarily private".

The «action» stereotype is used to signal how an agent acts 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. A coherency rule associated is: ''an attribute stereotyped with «action» is private and a method that is stereotyped using «action» is private and can only be called during the action phase of an agent".

The «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. A coherency rule associated is: ''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".

The «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 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. Coherency rules associated are:

Published in F. Bergenti, M.-P. Gleizes, and F. Zambonelli, editors, *Methodologies and Software Engineering for Agent Systems*, pages 157-176. Kluwer Publishing, 2004.

- 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».

The «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 or alteration. Representations that may evolve can be expressed using a multi-agent system. Coherency rules associated are:

- 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.

The «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»). A coherency rule associated is: A method stereotyped with «interaction» can only call methods stereotyped with «skill» or «interaction». Moreover, all the methods appearing on protocol diagrams are automatically stereotyped «interaction».

The «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»,

Published in F. Bergenti, M.-P. Gleizes, and F. Zambonelli, editors, *Methodologies and Software Engineering for Agent Systems*, pages 157-176. Kluwer Publishing, 2004.

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

An associated coherency rule is: ''a method stereotyped with «cooperation» is private".

Define Non Cooperative Situations. This step represents another contribution of ADELFE to the design workflow. Rules which allow the agent to have a cooperative attitude have to be defined: how to detect and to remove NCS in order to be more cooperative. During it, designers must fill up a table describing each NCS encountered by each previously identified agent.This table contains:

- The name of the NCS,
- The state in which the agent is during the detection of NCS. This state can be defined by a set of values of attributes or results of methods which can be stereotyped as «perception», «characteristic» or «representation»,
- The textual description of the NCS,
- The conditions describe the different elements that enable to locally detect the NCS. Methods and attributes used to express conditions must be stereotyped as «perception» or «representation» or «skill».
- The actions linked to the NCS. The actions describe what the agent has to do to remove this NCS. Methods and attributes used to express actions must be stereotyped «action».

For each table, at least one «cooperation»-stereotyped method must 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 as «perception», «representation» or «characteristic».

If several actions are possible to remove the detected NCS, you must define another method to choose the action to do. This method is stereotyped as «cooperation». If only one action is possible the definition of this second method is useless: this action will be always executed. These methods will be integrated in the behavior of the agent.

For instance, the NCS for a BookingAgent are:

- *Partnership incompetence*: the BA meets another BA that may be an uninteresting partner;
- *Booking incompetence*: the BA is in a cell that is uninteresting to book;
- *Message unproductiveness*: the BA receives a message that is not correctly addressed;

Published in F. Bergenti, M.-P. Gleizes, and F. Zambonelli, editors, *Methodologies and Software Engineering for Agent Systems*, pages 157-176. Kluwer Publishing, 2004.

- *Partnership conflict*: the BA meets another BA that is interesting, but this other BA has already a partner;

- *Booking conflict*: the BA is in a cell that is interesting to book but this cell is already booked;

- *Booking uselessness*: the BA meets its partner: they must separate to explore more efficiently the grid.

7. ADELFE Tools

Within ADELFE, three tools are associated with the process and the UML / AUML notations. The first tool is based on the OpenTool commercial software, enriched to take into account adaptive multi-agent system development. The second tool is an interactive tool which supports the process and helps designers to follow the process and to execute associated tasks. The last tool is a support decision tool to help designers to decide if the AMAS theory is relevant for the current system to design. In this section we only present the two first tools.

7.1 OpenTool for ADELFE

OpenTool is a development tool, written in the OTScript language, which is designed and distributed by TNI-Valiosys, one of the ADELFE partners. On the one hand, OpenTool is a commercialized graphical tool like Rational Rose and supports the UML notation to model applications while assuring that the produced models are valid. More specifically, it focuses on analysis and design of software written in Java. On the other hand, OpenTool enables meta-modeling in order to design specific configurations. This latter feature has been used to extend OpenTool to take into account the specificities of adaptive multi-agent systems and thus include them into ADELFE.

The first modification added to OpenTool concerns the static view of the model: the class diagram. Nine stereotypes are integrated to modify the semantics of classes and features depending on the specificities of cooperative agents (see §6.2).

As ADELFE reuses AUML to model interactions between agents, OpenTool was enhanced to construct AIP diagrams (see §6.1). AIP diagrams are an extension to existing UML sequence diagrams that enables different message sending cardinalities (AND, OR or XOR). This second modification was enriched with the possibility to easily attach a protocol to an agent class. Moreover, in order to simulate agents' behaviors by using finite state machines, OpenTool can automatically generate state-chart diagrams corresponding to protocols and roles within these protocols.

Published in F. Bergenti, M.-P. Gleizes, and F. Zambonelli, editors, *Methodologies and Software Engineering for Agent Systems*, pages 157-176. Kluwer Publishing, 2004.

7.2 Interactive Tools

ADELFE also provides an interactive tool that helps designers when following the process established in the method [1]. In classical object-oriented or agent-oriented methods, this kind of tool does not really exist. Even if some tools linked with agent-oriented methods exist (e.g. AgentTool [9] for MaSE, PTK for PASSI [8] or the INGENIAS tool [10]), they are not really a guide and a verification tool for designers following a methodological process. Generally, some guides like books or html texts are given (e.g., a guide to follow the RUP is available on the web site of Rational Software) but they are not really interactive tools able to follow a project through the different activities of the process. The ADELFE interactive tool is linked both with a tool to verify the AMAS adequacy and with OpenTool. It can communicate with OpenTool in order to access to different diagrams as process progresses. For these two reasons, it can be considered as a real guide that supports the notation adopted by the process and verifies the project consistency.

Each activity or step of the process is described by this tool and exemplified by applying it to the ETTO problem. Within the textual description or the example, some AMAS theory specific terms can be attached to a glossary in order to be explained. That is why the interactive tool is composed of several interfaces. The “Manager” interface indicates for the different opened projects, the different activities and steps designers have to follow when applying the methodology. The “Work Product” interface lists the work products that have been produced or that have to be produced yet regarding the progress when applying the methodology. The “Description” interface explains the different stages (activities or steps) designers must follow to apply the methodology process. The “Example” interface shows how the current stage has been applied to ETTO. The optional “Synthesis” interface shows a global view and an abstract of the already made activities. And finally the optional “Glossary” interface explains the terms used in the methodology and defines the stereotypes that have been added to UML.

8. Comparison with other Methodologies

ADELFE is based on object-oriented methodologies, follows the RUP (Rational Unified Process) and uses UML / AUML notations as MESSAGE [4]. It covers the entire process of software engineering like MESSAGE, PASSI and TROPOS [7]. And as DESIRE [3], MASSIVE, INGENIAS/MESSAGE [10], MaSE [9], PASSI, PROMETHEUS, it provides modeling graphical notations which are supported by tools. ADELFE is not a general method such as GAIA [15] but it has a niche, which concerns applications that require adaptive multi-agent system design using the AMAS theory. Therefore, like MESSAGE dedicated to telecoms applications, ADELFE gives guidelines for the identifi-

Published in F. Bergenti, M.-P. Gleizes, and F. Zambonelli, editors, *Methodologies and Software Engineering for Agent Systems*, pages 157-176. Kluwer Publishing, 2004.

cation of the application areas for which adaptive systems technology is better suited than other technologies e.g. object-oriented technologies.

Many methods, like AAII [12], TROPOS, MaSE or MESSAGE, do not focus on the dynamic aspect of the software environment and on the adaptation abilities of the software. TROPOS, like ADELFE, is concerned by dynamics. It expresses the dynamics and openness of the application in the requirements phases with the model of the environment and with particular soft goals. However, it does not give guidelines to design the right agents' behavior allowing the adaptability of the system.

In adaptive multi-agent systems, the environment (in which the system is operating) is a key notion; but in a general way, the environment modeling is not a central point in existing methodologies. In DESIRE, the environment is taken into account at the agent level in the "world interaction management module": an agent maintains and interacts with its environment in the same way as with other agents. In TROPOS, the environment model is described in terms of actors, their goals and interdependencies. In MESSAGE, the domain model captures some entities of the system environment and the interactions with the environment are described for each role in terms of sensory inputs and acquaintances, resources ownership and accesses, and finally tasks and actions. In AAII, the relation between the agent and the environment is taken into account in the interaction model.

At the design level, some methodologies are dedicated to an agent architecture as AAII with BDI, ADELFE with cooperative agents. In other methodologies such as GAIA, MESSAGE, TROPOS, the architecture of the implemented agents is not defined and it is quite open. Note that TROPOS offers different architecture styles (flat structure, pyramid, etc.) for its architectural design phase.

In the analysis workflow of GAIA, the agents are already identified and the methodology provides nothing to realize this identification. In TROPOS the agents are found inside the actors' set. In AAII, the elaboration and refinement of the agent model and the interaction model help the designer to define agents. The agent definition, which is given in MESSAGE and in ADELFE, defines the features that will be ascribed to the entities that the developer will choose to consider as agents.

9. Conclusion

The aim of this paper was to present the ADELFE methodology which is a multi-agent-oriented methodology suited to adaptive multi-agent systems based on the AMAS theory. ADELFE provides a new methodology to design a society of agents exhibiting a coherent activity. The first prototype is now operational and can be tested on the site http://www.irit.fr/ADELFE. Until now

Published in F. Bergenti, M.-P. Gleizes, and F. Zambonelli, editors, *Methodologies and Software Engineering for Agent Systems*, pages 157-176. Kluwer Publishing, 2004.

ADELFE has been used or is used in several case studies: an intranet system design, a timetabling problem, a flood forecast system (in progress), a mechanical design system (in progress) and a bioinformatics system (in progress).

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., the IRIT software engineering team, Carole Bernon and Valérie Camps.

References

[1] C. Bernon, V. Camps, M.-P. Gleizes, and G. Picard. Tools for self-organizing applications engineering. In G. Di Marzo Serugendo, A. Karageorgos, O.F. Rana, and F. Zambonelli, editors, *First International Workshop on Engineering Self-Organizing Applications (ESOA) at the Second International Joint Conference on Autonomous Agents and Multi-Agents Systems (AAMAS'03)*, Melbourne, Australia, July 2003.

[2] C. Bernon, M.-P. Gleizes, S. Peyruqueou, and G. Picard. Adelfe: a methodology for adaptive multi-agent systems engineering. In P. Petta, R. Tolksdorf, and F. Zambonelli, editors, *Third International Workshop on Engineering Societies in the Agents World (ESAW-2002)*, volume 2577, pages 156–169, Madrid, Spain, September 2002. Springer-Verlag (LNAI).

[3] F. Brazier, C. Jonker, and J. Treur. Compositional design and reuse of a generic agent model. *International Journal of Cooperative Information Systems*, 9(3):171–207, 2000.

[4] G. Caire, W. Coulier, F. Garijo, J. Gomez, J. Pavon, F. Leal, P. Chainho, P. Kearney, J. Stark, R. Evans, and P. Massonet. Agent oriented analysis using message/uml. In M.J. Wooldridge, G. WeiSS, and P. Ciancarini, editors, *Agent-Oriented Software Engineering II, Second International Workshop, AOSE 2001*, volume 2222 of *LNCS*, pages 119–135, Montreal, Canada, May 29th 2001. Springer-Verlag.

[5] D. Capera, JP. Georgé, M-P. Gleizes, and P. Glize. The amas theory for complex problem solving based on self-organizing cooperative agents. In *1st International workshop on Theory and Practice of Open Computational Systems (TAPOCS) at IEEE 12th International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2003)*, pages 383–388. IEEE Computer Society, 9-11 June 2003.

Published in F. Bergenti, M.-P. Gleizes, and F. Zambonelli, editors, *Methodologies and Software Engineering for Agent Systems*, pages 157-176. Kluwer Publishing, 2004.

[6] D. Capera, JP. Georgé, M-P. Gleizes, and P. Glize. Emergence of organisations, emergence of functions. In *FAISB'03 symposium on Adaptive Agents and Multi-Agent Systems*, April 2003.

[7] J. Castro, M. Kolp, and J. Mylopoulos. A requirements-driven development methodology. In K.R. Dittrich, A. Geppert, and M.C. Norrie, editors, *Proceedings of the 13th International Conference on Advanced Information Systems Engineering (CAiSE'01)*, volume 2068, pages 108–123, Interlaken, Switzerland, June 2001. Springer-Verlag (LNCS).

[8] M. Cossentino. Different perspectives in designing multi-agent system. In *Designing Multi-Agent System, AgeS'02 (Agent Technology and Software Engineering) Workshop at NodE'02*, Erfurt, Germany, October 2001.

[9] S.A. DeLoach and M. Wood. Developing multiagent systems with agenttool. In C. Castelfranchi and Y. Lesperance, editors, *Intelligent Agents VII. AgentTheories Architectures and Languages, 7th International Workshop (ATAL 2000)*, volume 1986 of *LNCS*, pages 46–60, Boston, MA, USA, July 7–9 2001. Springer-Verlag.

[10] J. Gomez Sanz and R. Fuentes. Agent oriented system engineering with ingenias. In *Fourth Iberoamerican Workshop on Multi-Agent Systems, Iberagents'02*, 2002.

[11] I. Jacobson, G. Booch, and J. Rumbaugh. *The Unified Software Development Process*. Addison-Wesley, 1999.

[12] D. Kinny, M. Georgeff, and A. Rao. A methodology and modelling technique for systems of bdi agents. In W. Van de Velde and J. W. Perram, editors, *Agents Breaking Away: Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a MultiAgent World*, volume 1038 of *LNAI*, pages 51–71. Springer-Verlag, 1996.

[13] J. Odell, H.V. Parunak, and B. Bauer. Extending uml for agents. In *Proceedings of the Agent Oriented Information Systems (AOIS) Workshop at the 17th National Conference on Artificial Intelligence (AAAI)*, 2000.

[14] S. Russel and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Series, 1995.

[15] M. Wooldridge, N.R. Jennings, and D. Kinny. A methodology for agent-oriented analysis and design. In Oren Etzioni, Jörg P. Müller, and Jeffrey M. Bradshaw, editors, *Proceedings of the 3rd International Conference on Autonomous Agents (Agents 99)*, pages 69–76, Seattle, WA, May 1999. ACM Press.