UER Technique: Conceptualisation for Agent Oriented Development

Carlos A. Iglesias
Departamento Ingeniería de Sistemas Telemáticos, Universidad Politécnica de Madrid
E.T.S.I. Telecomunicación, Ciudad Universitaria s/n, 28040 Madrid (Spain)

And

Mercedes Garijo
Departamento Ingeniería de Sistemas Telemáticos, Universidad Politécnica de Madrid
E.T.S.I. Telecomunicación, Ciudad Universitaria s/n, 28040 Madrid (Spain)

Abstract

The problem of conceptualisation is the first step towards the identification of the functional requirements of a system. This article proposes two extensions of well-known object oriented techniques: UER (User-Environment-Responsibility) technique and enhanced CRC (Class-Responsibility-Collaboration) cards. UER technique consists of (a) looking for the users of systems and describing the ways the system is used; (b) looking for the objects of the environment and describing the possible interactions; and (c) looking for the general requirements or goals of the system, the actions that it should carry out without explicit interaction. The enhanced CRC cards together with the internal use cases technique is used for defining collaborations between agents. These techniques can be easily integrated in UML (Unified Modelling Language) [2], defining the new notation symbols as stereotypes.

Keywords: Agent Oriented Software Engineering, multi-agent systems modelling, autonomous agents modelling, environment cases, goal cases

1. Introduction

The problem of conceptualisation is the first step towards the identification of the functional requirements of a system. One of the most extended techniques for getting a first idea of the system is the *Use Case* technique [5]. The technique consists in identifying the possible users of the systems, and the possible user goals, describing ways of achieving these user goals, that are called *use cases*. Usually, different use cases can be combined with the relationships *extends* (if a use case is an extension of another one) or *uses* (if a use case is a part of another one). This technique is very simple and intuitive and has been very successful for requirements elicitation and validation.

This technique can be used for conceptualising a multiagent system, as described in [4]. Nevertheless, autonomous agents are distinguished because they do not need a user that supervises their execution. So, while with *use cases* we have to answer the question "*How is used my system?*", we could ask ourselves for other requirements of our system such as: "*When and how my system act and react to the environment?*" (*environment cases*) and "*What are the goals of the system?*" (*responsibility or goal cases*). This article introduces these new concepts in the conceptualisation phase and the corresponding techniques and notations.

In order to conceptualise an agent-based system, two general techniques are proposed: the new UER cases technique (section 2), that deals with the identification of use, reaction and goal cases of an agent or a multiagent system, and the enhanced Class-Collaboration-Responsibility Cards technique (section 3) that deals with the identification of responsibilities, plans and collaborations of an agent. Both techniques are complementary. the UER technique can be used for both autonomous or multiagent systems (for identifying use, reactive and goal cases of the whole system). The enhanced CRC cards are only used for conceptualising multiagent systems, since they guide the definition of collaborative scenarios.

2. UER technique

The UER (User-Environment-Responsibility) technique proposes the combination of user, environment and responsibility-driven analysis for conceptualising a system from an agent-oriented perspective. This technique can be used for conceptualising a particular autonomous agent or the general requirements of a multiagent system.

User-Centered Analysis. The potential users (called *actors*) of the system are identified, together with their possible tasks or functions. The result of this analysis is the set of *use cases*. This analysis answers the question: How are the possible uses of the multiagent system?

Environment-Centered Analysis. Agents can be situated in an environment, and this environment needs to be modelled. In particular, we are interested in modelling how the system can act and react to this environment. The result of this analysis is the set of *reaction cases*. This analysis answers the question: How the multiagent system has to react to the environment?

Responsibility-driven Analysis. In contrast with usual software systems, multiagent systems can act proactively. The user can desire that the system has some responsibilities, that is, the user can assign some goals or responsibilities to the system and the system carries out these responsibilities without a direct demand. This analysis answers the question: What are the goals of the system? The main difference of *goal cases* from the user cases, is that the uses cases show how the system gives an answer to a user request, while the goal cases show how the system behaves when some condition is fulfilled.

User-Centered Analysis

A *use case* [5], [6], [7] describes the possible interactions or uses of a user with the system. System users are called *actors*, and represent external entities of the system. Use cases can be combined, pointing out if a use case extends or uses a previous use case.

User-Centered Analysis consists of the following steps [5], [6], [8], [7]:

• Identify the actors. It is specially relevant to identify the roles played by the actors. Each role is considered a different actor. There are two general kinds of actors: human actors (round head) and software actors (square head), as shown in Fig. 1.[1]

• Identify the use cases. This process can be carried out by answering the following questions [5], [7]:

– What are the main tasks or functions carried out by each actor?

– What system information is acquired, produced or changed by each actor?

– Does any actor inform about external changes in the system environment?

– What information is needed by each system actor?

– Does any actor desire to be informed about unexpected changes?

• Group the use cases if they are variations of the same subject (for example, 'move a heavy stone', 'move a light stone').

• Determine the interactions of each identified use case.

• Describe the use cases, using both a graphical notation [3], [2], [7] and textual templates.

• Consider every possible exception that can happen during the interactions and how this affects to the use cases.

• Look for relationships among the use cases: extract common parts and point out if a use case adds the interactions of another use case (relationship "uses") or adds information contained in another use case (relationship "extends" or "includes"). A use case can also inherit the general interaction of an abstract use case with the 'relationship "instantiates".

• Describe the interactions of each scenario, using MSC (Message Sequence Chart) notation [3]. MSC has been selected because is a standardised formal description technique with a textual and graphical grammar. Some of the relevant features for our purposes are the availability of a language (HMSC, High Level MSC) for defining the phases of the interaction, and the definition of operators for expressing alternatives, exceptions and concurrence in the same diagram. Sequence and collaboration diagrams do not allow to express these issues in such an easy way, but can also be used.

Environment Centered Analysis

The goal of environment centered analysis is to identify the relevant objects of the environment and the possible actions and reactions of the agent. This will be later used for agent sensor modelling.

Environment Centered Analysis consists of the following steps:

• Identify objects of the environment. This objects are shown in the use case diagram as clouds (Fig. 1). This symbol can be defined in UML as an stereotype.

• Identify the possible events coming from each objects and establish a hierarchy if possible.

• Identify the possible actions each agent can carry out on the environment objects.

• Describe (in natural language) the reaction cases coming from interaction with the environment. Describe in detail each possible scenario. Think if there are several scenarios coming from the same reaction case, and if every scenario is autonomous (it is only managed by the agent that receives the stimuli) or cooperative (it is managed in cooperation with other agents).

• Group related reactive cases with the relationships "uses", "extends", "includes" or "instantiates". For example, "avoid obstacle" can group different scenarios for avoiding an obstacle depending on its nature, and can be avoided in an autonomous way (e.g. just going to the left of the obstacle) or in a cooperative way (e.g. asking for help to move it).

• Describe the reactive goal: its name, the activation condition (e.g. a wall very close), the deactivation condition and the successful and failure condition (when the reaction has been effective or not).

Goal Driven Analysis

Goal driven analysis deals with the definition of requirements of the system, that should be fulfilled without the direct interaction with the user.

Goal Driven Analysis consists of the following steps:

• Identify responsibilities (goals) of the system that

[1]This distinction is used for describing later the interactions, using an agent communication language based on speech acts or not.

require some action. Some of the hints for identifying these goals are:

– Look for non functional requirements, such as time requirements (e.g. 'Give an answer before 5 minutes') or security requirements (e.g. 'Buy a product in a secure way'). Sometimes the agent needs to carry out special actions to achieve these goals.

– Describe when some internal variable of the agent can reach a not desired value and some action should be carried out. For example, high/low temperature, too many processes, etc.).

– Describe undesired states, possible failures of the system that should require action to be avoided.

• Describe the proactive goal: its name, its type (persistent, priority, etc.), the activation condition (e.g. no fuel or idle), the deactivation condition and the successful and failure condition (when the plan has been effective or not).

• Group related goals using the relationships "uses", "extends", "includes" or "instantiates".

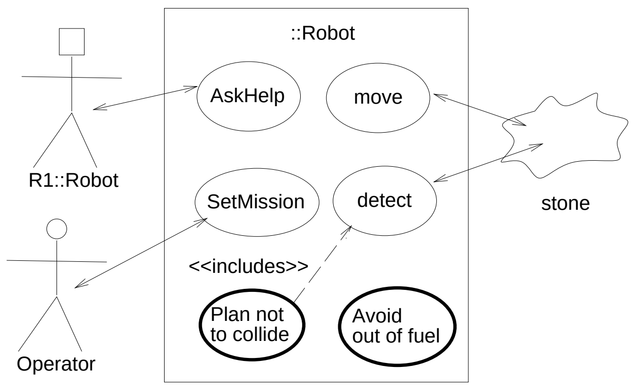

Fig. 1. Robot UER cases

Example

In order to illustrate the technique, we can consider a 'robot world' where there are a set of robots that transport boxes from one place to another. The robots need fuel to run and should not be blocked by stones. Human operators can order the robots to move one box from an origin to a destination or to stop a task.

User-Centered Analysis. In this case, the following actors (Fig. 1) can be identified: another *robot* that helps the robot to move a heavy box and a human *operator* that sets the mission of the robot (e.g. move a box from one position to a destination).

After some analysis, the use case *SetMission* can be further refined as shown in Fig. 2. The general case *SetMission* consists of asking for a mission, that is accepted or refused. *SetMission* is a generalisation of the possible missions: *TransportBox* and *CountBoxes.* The use case *TransportBox* can be included by the case *FindAndTransportBox*, where the user just establishes the number of a particular box and where it should be delivered.

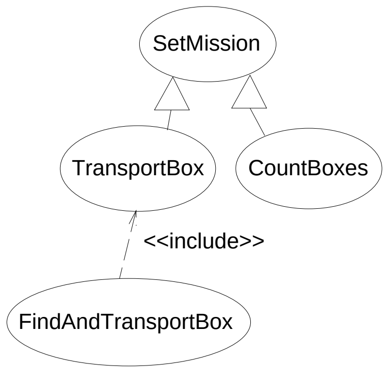

Fig. 2. Relationship between use cases

Once the actors and use cases have been determined, they are described with textual templates and MSCs as a graphical notation. For example, Fig. 3 shows the interactions of the use case *SetMission.* The operator requests to carry out a mission and the robot can give two alternative answers, to accept or to refuse the mission. The messages of the MSC follow the syntax *<speech-act> (<content>).*

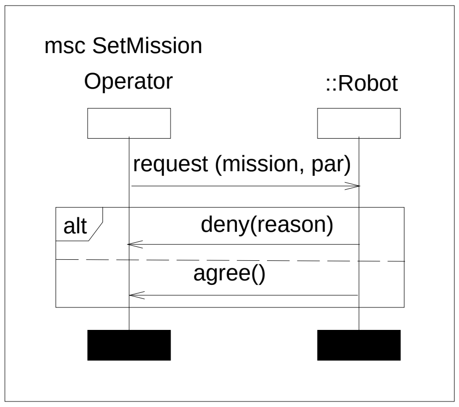

Fig. 3. Scenario of the use case

Environment-Centered Analysis One environment object can be identified: the *stone*, that the agent detect, count and can move. The relevant attributes of the stone for the agent are its position, its number and its weight. The agent needs sensors to detect that there is a stone and not to collide with it, so this is the main reaction case: *Detect.* When an obstacle is detected, the reaction case the agent should try not to collide (reaction case *Avoid Collision*), that includes the detection of the stone. Several scenarios can be thought for avoiding the collision: the agent decides to avoid the obstacle, for example going to the left (reaction case *AvoidObstacle*), decides to stop because there is no way out (reaction case *Stop*), decides to move alone the obstacle (reaction case *MoveAlone*) or decides to ask for help to other agent to move the box (reaction case *AskHelp*[2].

[2]As the reader probably has observed, *AskHelp* was previously defined

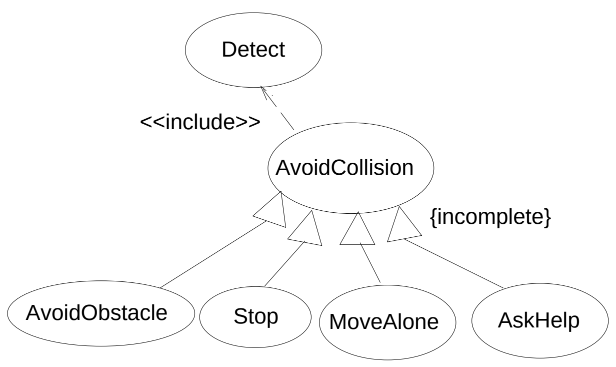

Fig. 4. Relationship between reaction cases

Goal Driven Analysis

Two requirements (goals) have been identified: the robot should not collide with the stones or robots and the robot should not get out of fuel.

The first requirement (goal case *Plan not to collide*) is similar to the reaction case *AvoidObstacle*. The difference is that we want that the robot try to avoid an obstacle even when it does not receive any stimuli from the sensors. For example, if the robot can select two alternative paths to transport a box, the selection can take into account the position of the rest of the robots that could eventually collide with it.

The second requirement (goal case *Avoid out of fuel*) can be further refined as shown in Fig. 5. This goal case can be activated when an indicator of out of fuel is activated, being a kind of "internal reaction case". Another possible scenario is that the robot can go to the fuel station when is idle to avoid being out of fuel (goal case*Go Station if idle*).

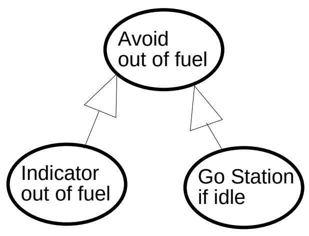

Fig. 5. Relationship between goal cases

After identifying the cases, it is needed to describe the different scenarios and attributes of each case. For example, the goal *Go Station if idle* is a persistent goal whose activation condition is being low of fuel and being idle. The deactivation condition can be to receive an order while achieving the goal. The goal is achieved if the tank of fuel is filled and fails in other case. Another alternative could be to maintain this goal active even if there is an ongoing order but the fuel station is close. These two policies need to be tested on the environment.

as a use case; but the roles are interchanged. As a use case, the robot received the petition and now it is the initiator of the petition.

3. Enhanced CRC cards and internal use cases

The well known CRC (Class Responsibility Collaboration) cards [1], [9] technique provides a method for organising the relevant classes for modelling a system. This technique was initially used [1] for teaching object fundamentals in a collaborative environment. The technique consists of filling cards. Each card has a class name and two columns. The left column shows the responsibilities of the class, that are the tasks the class can perform or knowledge it has, and the right column show the classes that collaborate to achieve these tasks or obtain this knowledge.

This technique can be easily modified from an agent perspective. A CRC is filled for each agent role, describing its class. Each CRC is divided into five columns (table I): goals assigned, plans for achieving these goals, knowledge needed to carry out the plans, collaborators in these plans, and services used in the collaboration. The back side of the CRC is used for annotations or extended description of the front side.

Internal use cases are also based on RDD [9] and its CRC (Class Responsibility Collaboration) cards. Taking as input the use cases of the conceptualisation phase and some initial agents, we can think that each agent "uses" other agent(s), and can use these agents with different roles. We look for such an agent in our agent-library for reusing, combining in this way the top-down and bottom-up approach. The external use cases coming from the actors of the multiagent system are decomposed in use cases that are assigned to agent roles of the system.

4. Conclusions

This article has proposed several techniques that can be used for conceptualising a system from an agent perspective.

UER technique considers three perspectives for conceiving the system: studying the 'uses' of external actors, studying the interactions with the objects of the environment, and studying the responsibilities or goals of the system. This technique can be used for conceiving a particular agent or the requirements of a multiagent system.

This article also proposes an agent-oriented version of CRC cards that can be used in conjunction with the use cases techniques. These techniques deal with finding collaborations between agents in a multiagent system and provide a method for agent reusability.

These techniques have the advantage of being easily integrated in the current object-oriented CASE tools using UML and are integrated in a wider agent-oriented methodology called *MAS-CommonKADS* [3].

Agent: Robot1			Class: Robot	
Goals	Plans	Knowledge	Collaborator	Service
Maintain fuel	Go FS if idle	FS Ontology	FS Clerk	Ask For Fuel
	Go FS if Fuel Indicator On	FS Ontology	FS Clerk	Ask For Fuel

TABLE I

Example of Agent oriented CRC Card. FS: Fuel-Station

5. Acknowledgements

This research is funded in part by the Spanish Government under the CICYT projects TIC91-0107 and TIC94-0139.

The authors would like to thank the students that are applying the methodology for many suggestions and comments.

6. References

[1] Kent Beck and Ward Cunningham. A laboratory for teaching object-oriented thinking. In *OOPSLA'89 Conference Proceedings*, New Orleans, Louisiana, USA, October 1989.

[2] Ivar Jacobson Grady Booch, James Rumbaugh. *The Unified Modeling Language User Guide*. Addison Wesley, 1998.

[3] Carlos A. Iglesias, Mercedes Garijo, José C. González, and Juan R. Velasco. Analysis and design of multiagent systems using MAS-CommonKADS. In AAAI'97 Workshop on Agent Theories, Architectures and Languages, Providence, RI, July 1997. ATAL. (An extended version of this paper has been published in *INTELLIGENT AGENTS IV: Agent Theories, Architectures, and Languages*, Springer Verlag, 1998.

[4] Carlos A. Iglesias, Mercedes Garijo, José C. González, and Juan R. Velasco. Analysis and design of multiagent systems using MAS-CommonKADS. In M. Wooldridge, M. Singh, and A. Rao, editors, INTELLIGENT AGENTS IV: Agent Theories, Architectures, and Languages, volume 1365, pages 313–329. Springer-Verlag, 1998. (A reduced version of this paper has been published in *AAAI'97 Workshop on Agent Theories, Architectures and Languages.*

[5] I. Jacobson, M. Christerson, P. Jonsson, and G. Övergaard. *Object-Oriented Software Engineering. A Use Case Driven Approach.* ACM Press, 1992.

[6] Roger S. Pressman. *Software Engineering: A Practitioner's Approach.* McGraw-Hill, 4th edition, 1996.

[7] James Rumbaugh. Getting started. using use cases to capture requirements. *JOOP Journal of Object Oriented Programming*, pages 8–23, September 1994.

[8] James Rumbaugh. OMT: The development model. *JOOP Journal of Object Oriented Programming*, pages 8–16, 76, May 1995.

[9] R. Wirfs-Brock, B. Wilkerson, and L. Wiener. *Designing Object-Oriented Software.* Prentice-Hall, 1990.