Applying UML Use Case Diagrams to Agents Representation

Antonio Chella
Dip. di Automatica ed Informatica
University of Palermo
Viale delle Scienze, Palermo, Italy
chella@unipa.it

Massimo Cossentino
Dip. di Ingegneria Elettrica
University of Palermo
Viale delle Scienze, Palermo, Italy
maxco@unipa.it

Umberto Lo Faso
Dip. di Automatica ed Informatica
University of Palermo
Viale delle Scienze, Palermo, Italy
lofaso@unipa.it

Abstract

This paper is the first step of an exploration of the possibilities offered by the application of UML to agent-based software. Starting from UML definitions of use cases and actors we discuss their correspondence to agents and external entities of the environment. An example is also presented that starts form a classical well–known case of study.

1. Introduction

Software agents are always increasing in importance and consequently in their complexity.

This growth, as it seems to us, hasn't been supported by a consequent increase of the design and developing techniques that, conversely, well support the design of conventional object-oriented software.

In the past several authors proved to create a robotics (or agent) dedicated design method or language [1], [2], [3], [4], [8], [9].

In this paper, we begin to face the problem of combining both agents and software engineering approaches and, in this context, we believe that UML can be usefully applied in order to design an agent-based software. Moreover we think that this choice is also a suitable one because UML is going to become, by now, a well-established industrial standard.

The first step in our exploration of this approach is the evaluation of the possibility of applying the use case diagrams to higher level descriptions of agents and of their interactions.

We also apply our idea to a very simple example that shows the strength and clearness of our approach.

In the next future, we are going to experiment this method in the programming of a real RWI B21 robot equipped with sonar sensors, stereo vision system and laser scanner.

This work will give us also a chance to pursue a certain degree of portability of the software design from the B21 robot to other ones which need similar behaviours.

The following pages are a brief introduction to our idea. To be specific: in the second chapter we give a short description of the Unified Modeling Language (UML); in the third chapter, starting from definitions of Use Case and Agent, we identify their similarities and the way of using use cases, to describe agents in the use-case driven analysis; in the fourth chapter an example is provided.

2. The Unified Modeling Language

UML is a language that can be used to analyse, specify, construct and document a software artefact. Its origins come from several object-oriented modelling approaches (and languages) of some years ago (Booch's OOADA, Jacobson's OOSE, Rumbaugh OMT and others): UML is object-oriented itself. In the late 1997 UML 1.1 was approved and adopted by OMG (Object Management Group). At this date UML became the first standard modelling language and the most supported by industry. The greatest names in software production have given their contribution to the definition of this standard and are now interested in his growth. The academic world is involved in the exploration of the various problems connected with the introduction of UML in the software design and development process. UML is not thought to support a well specified

design process, it can be used to support new approaches as well as older ones. This is a great field of interest and research that is still opened.

In this context we have studied the application of UML to agent-based software, looking both at the well-known traditional approaches (trying to reconcile these ones with the new design language) and at the prospect of developing new specific methods.

2.1. UML diagrams

During the analysis phase, with special regard to the problem of describing a system to be developed, its context, the interaction of the external entities, UML offers two diagrams: the Use Case Diagram and the traditional, well known, Class Diagram. The first diagram is centred upon use cases (illustrating the functional aspects of the system) and actors (entities external to the system and interacting with it). The class diagram contains both classes (entities of the system) and their relationships. Classes could contain attributes and methods (addressing the behaviour of the system).

Looking at the software system, the UML elements can be grouped in a few conceptual areas: static structure, dynamic behaviour, implementation constructs, model organisation, extensibility mechanisms.

To describe the static structure of our application we can use the class diagram.

The system's dynamic behaviour can be described using collaboration or sequence diagram. These are different points of view of the same scenario. Scenarios are paths around use cases illustrating one of the possible behaviour of the software. In collaboration diagrams attention is focused upon what message a certain entity of the system exchanges with the other ones. Sequence diagrams again show messages but arranged in time order.

Finite state diagrams and activity diagrams can also be used to describe a certain procedure or the life of a class.

To support the implementation aspects of design, UML offers component and deployment diagrams. In the component diagrams classes are associated with components (executables, libraries, ...) that will be created with their relations. In deployment diagrams processes and nodes (execution units or other devices) are shown with their connections.

Packages can be used to organise the structure of large systems: they allow simplifying the management of systems which involve a great number of elements by organising them in an hierarchical structure.

Several extensibility mechanisms are part of UML: stereotypes (model elements similar to standard ones but with additional constraints), tagged values (that can be attached to any model element to contain additional information) and constraints (that can be used to create semantic relationship among model elements that specifies Boolean conditions and propositions). Constraints are described in words and are attached to a model element: a specific language, named OCL (object constraint language), has been associated to UML in order to support constraints.

3. Use cases and agents

In this chapter we want to show that it is possible to use a use case diagram to describe an agent system from the social level point of view.

Use case diagrams "show actors and use cases together with their relationships"[11].

We would like to demonstrate that they could represent agents (use cases), environment (actors) and interactions (relationships).

Before going into our argument, a definition of what we define as an agent can be helpful.

"An agent is an encapsulated computer system that is situated in some environment and that is capable of flexible, autonomous action in that environment in order to meet its design objectives"[10].

We want to put this agent in correspondence with a use case.

According to UML standards, the use case definition is:

"A use case is a coherent unit of functionality provided by a system, a subsystem or a class as manifested by sequences of messages exchanged among the system and one or more outside interactors (called actors) together with actions performed by the system."

We can find many contact points between these definitions.

If the use case is 'a coherent unit of functionality' it can describe the 'flexible, autonomous action' of an agent. As a consequence, we can represent an agent and his vocational behaviour by a use case.

The behaviour of an agent is the consequence of his interaction with the real world (or other agents), and of his own purposes ('its design objectives').

The interactions between agents in our view can be seen as "messages exchanged" in the previous definition. These messages are

transmitted through the relationships that exist among the system elements.

UML comprehends only 4 standard kinds of relationships: associate, extend, generalize and include. Other new relationships can be defined by the user and in so doing every kind of interaction can be represented.

According to UML definition, the actor "defines a coherent set of roles that the users of an entity can play when interacting with the entity."

Starting from this definition we can think that the environment, in his interaction with the agent, plays the role of an actor (represented with a use case). For example the environment can give to an agent the motivation to perform (or not) a specific behaviour. Other agents from a certain point of view (according to the designer perspective) can be seen as actors (i.e. entities external to the system to be designed) or use cases (entities internal to the system to be designed).

It is possible to think that in the previous actor's definition, the role of "active subject" is assigned to the actor. In the agent representation we propose it isn't so. Indeed, our choice of representing entities external to our agent system as actors seems to be in accordance with "Actors model parties outside an entity, …" and "Since an actor is outside the entity, its internal structure is not defined but only its external view as seen from the entity" [11] (this question also recalls the problem of action and intention as discussed in [12], [13]).

Use case diagrams in the illustrated approach, can describe agents together with their interactions and relations. Use cases diagrams illustrate also the agents' purposes from a static, structural point of view (external, not of implementation). These diagrams can't obviously describe the agents' dynamic behaviour, which can be realized in the countless possible scenarios of a complex agent system.

Other UML diagrams such as Activity diagrams and Sequence (or Collaboration) diagrams can describe more precisely the dynamic behaviour of an agent society.

4. Example

Consider the well known example of the "foraging" robot [1]. Its behaviour can be represented through the FSA (Augmented Finite State) diagram of fig.1.

The robot explores the environment around him until it detects some food. Then he directs towards the food, grabs it and delivers it to a designed site.

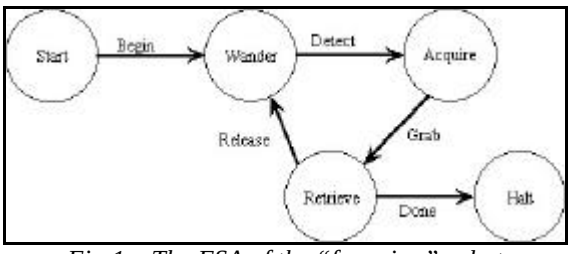

Fig.1 – The FSA of the "foraging" robot

This diagram illustrates the dynamics of the behaviour of the robot but it doesn't describe the 'operative situation'.

In making an agent-based software and specially in the relative analysis phase we could take benefits from a description of the "functionality of the model as perceived by outside users" [6] like that provided by an use case diagram.

In fig. 2 the "foraging" robot system is seen from the outside point of view of the use case diagram.

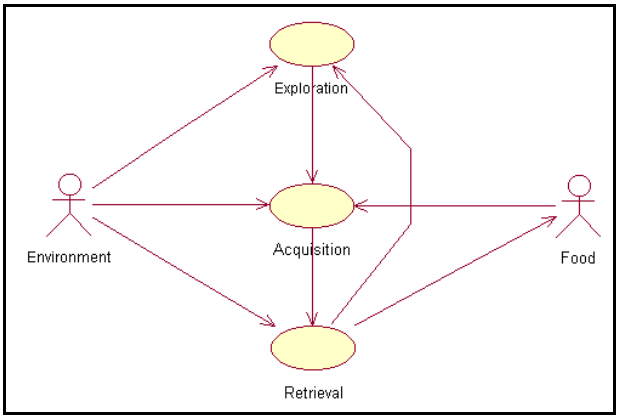

Fig.2 – Use case diagram of the "foraging robot"

Two actors (represented as a "stick" man) can be identified: the environment and the food; behaviours of fig.1 are represented as use cases (represented as ovals)

The environment 'communicates' the positions of the objects contained in the robot world (the robot receives this data through his own sensors). Using this information the robot walks around searching for food (use case 'exploration'). When he finds some food (use case 'acquisition') he moves towards it and collects it; in this use case the robot receives information from the environment (obstacle position) and from food (its position). Finally he brings to the selected site the collected food. In so doing the robot needs information from the environment to avoid obstacles and, obviously, uses the food that he is taking with him.

This use case diagram looks at the same operative situation of the FSA in fig.1 but from a

different point of view. Several agents (Exploration, Acquisition, Retrieval) are present and collaborate to achieve the goal (collecting food and taking it home); the communications between external entities (i.e. environment and food) and agents of the system are well specified (and can, eventually, be typed).

5. Conclusion

The introduction of use case diagrams in the design of agent based software creates the opportunity of enhancing the analysis phase using one of the most interesting software engineering approach to the problem.

These diagrams allow to use an high level of abstraction and can well describe a great population of agents, visualising their mutual communications.

This contribution represents only the first step of a study path in the direction of applying UML to agents. Several interesting problems can be identified, one of them is the analysis and design of concurrency.

REFERENCES

[1]. Albus, J., McCain, H. and Lumia, R.: NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM), NBS Technical Note 1235, Robot Systems Division, NIST, 1987.

[2]. Lyons, D. and Arbib, M.: A Formal Model of Computation for sensory-Based Robotics, IEEE Trans. on Robotics and Automation, vol. 6, no. 3, pp. 280-293, 1989.

[3]. Kaelbling, L. and Rosenschein, S.: Action and Planning in Embedded Agents, in: P.Maes (ed.): Designing Autonomous Agents, MIT Press, Cambridge, MA, pp. 35-48, 1991.

[4]. Brooks, R.: The Behavior Language, AI Memo 1227, MIT AI Labroatory, 1990.

[5]. R.C. Arkin – "Behavior-Based Robotics" – The MIT Press – Cambridge MA

[6]. "The Unified Modelling Language Reference Manual" – J. Rumbaugh, I. Jacobson, G. Booch – Addison Wesley

[7]. M. Wooldridge, N.R. Jennings, Intelligent agents: Theory and practice, Knowledge Engineering Review 10 (2) (1995) 115-152.

[8]. M. Wooldridge, N.R. Jennings, Software engineering with agents: pitfall and pratfalls, IEEE Internet Computing 3 (3) (1999) 20-27.

[9]. M. Wooldridge, Agent-based software engineering, IEEE Proc. Software Engineering 144 (1) (1997) 26-37

[10]. N.R. Jennings, On agent-based software engineering, Artificial Intelligence 117 (2000) 277-296.

[11]. OMG Unified Modeling Language – Ver. 1.3 – June 99

[12]. J. R. Searle, "Minds, brains and programs" in "The behavioural and Brain Sciences" – 1980 – Cambridge University Press.

[13]. D.R.Hofstadter, D.C.Dennett: "The Mind's I", Basic Books, 1981.