Using UML 2.1 to model Multi-Agent Systems

Darshan S. Dillon, Tharam S. Dillon, and Elizabeth Chang

Digital Ecosystems and Business Intelligence Institute,
Curtin University of Technology Perth, Australia
(Darshan.Dillon, Tharam.Dillon, Elizabeth.Chang)@cbs.curtin.edu.au

Abstract. The use of UML 2.1 to model a broad range of systems is evident from the variety of UML diagrams in academia and in the marketplace. One class of systems currently gaining popularity are Multi-Agent Systems. There are efforts underway to use UML to model these systems and these efforts are both productive and form the basis for both a methodology and a notation for systems of this type.

Acknowledgement. The authors would like to acknowledge the invaluable assistance and suggestions of Maja Hadzic as we authored this paper.

1 Introduction, Agents and their characteristics

In this paper we first introduce what an Agent is, the key characteristics of an Agent, the scope of this paper in terms of what we model in Multi-Agent Systems, and finally future directions.

In order to define what an agent is we should first consider a definition from the literature.

An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives.[1]

From this definition a number of points are clear. Firstly, the location of the computer program is important. This is so because the program can migrate from one machine to another. This is not the usual pattern of behaviour for computer programs. They usually are installed, configured and run on a particular machine. They do not *travel,* as such. Secondly, the computer program is capable of acting automously, which means it is not dependant on any other program. This goes together with the fact that agents are mobile. They can be launched by a user on a particular machine, and travel, severing their connection with the user and concentrating their state related information within themselves. Thirdly, the computer program is goal-driven and can choose to act in a way that satisfies agents design objectives. Most computer programs are data-driven, reacting to inputs.

Finally, agents play an important role in embedded and ubiqitious computer systems. They are particularly important in goal-oriented or mission-oriented environments. The modeling and design of agents is an important first step for the building of agent-based systems.

There are six key characteristics of an agent[2]. They are as follows.

1. Autonomous – That is, an agent can perform independently from other agents by making decisions based on the internal state of itself and information from the environment.

2. Sociable – That is, an agent can co-operate and collaborate with other agents by using a common language to communicate with each other.

3. Service Discovery – Agents are able to identify desired services.

4. Reactive – That is, an agent is pro-active. It can perform tasks that may be beneficial to the user even though it has not been explicitly asked to perform those tasks.

5. Mobility – Agents can move across networks from any location. They can be assigned a task and sent over the web after which their connection to the user can be severed. Their state can be centralized within themselves.

6. Goal-Driven Execution – Each Agent has a goal that is it constantly trying to meet.

2 Scope

As with any paper, we need to define the scope we will work within. In the case of this paper, we will seek to model how the sociable and goal-driven nature of agents can be expressed using UML 2.1.

3 Agent Characteristics Modeled

3.1 Modelling Sociable Characteristics of Agents

In modeling Agents, one of their key characteristics is that they are sociable. This means that they are able to interact with each other in order to co-operate, collaborate and negotiate with respect to information, knowledge and services. Very often each agent will have only part of the full picture needed to solve the problem at hand. The

ability to subdivide the tasks in order to reduce the complexity of the problem, have individual agents work only on their aspect of the problem, and then combine sub-solutions into a final solution is extremely helpful and productive.

In AgentUML, previous researchers have been modeling Agent protocols[5,6] using a non-standard version of sequence diagrams where each rectangle represents an agent playing a different role. We say non-standard because the rectangles at the head of lifelines are meant to represent classes, not agents. Having multiple rectangles each representing the same Agent is also non-standard, where each rectangle represents the Agent playing a particular role.

3.2 Modelling roles of Agents

Each agent is defined by specifying a specific set of roles that it plays. Each role could be associated with a distinct interface. These interfaces could be specified by a technique called method lifting outlined below. Method lifting defines a composite class. What are composite classes ? If we first consider a hierarchy of component classes, each of which has an interface. If we relate these component classes to a composite class that also has an interface, and which is formed by taking a selection of methods from the interfaces of the component classes. This process of relating the interface of component classes to the interface of a composite class is known as method lifting. In the example below, the methods A, B & C are individually chosen from different component classes and combined in the composite class at the head of the hierarchy. This is shown below.