Exploiting UML in the Design of Multi-Agent Systems

Federico Bergenti and Agostino Poggi

Dipartimento di Ingegneria dell'Informazione, Università degli Studi di Parma
Parco Area delle Scienze 181A, Parma, Italy
{Bergenti, Poggi}@CE.UniPR.IT

Abstract. Software engineering describes a system at different levels of abstraction. Agent-oriented software engineering introduces a new level, called the agent level, to allow the software architect modelling a system in terms of interacting agents. This level of abstraction is not yet supported by an accepted diagrammatic notation even if a number of proposals are available. This paper shows how UML can be exploited to design a multi-agent system at the agent level. In particular, it presents a set of agent-oriented diagrams intended to provide an UML-based notation to model: the architecture of the multi-agent system, the ontology followed by agents and the interaction protocols used to coordinate agents. The presented notation exploits stereotypes to associate an agent-oriented semantic with class and collaboration diagrams. Therefore, it can be managed with any off-the-shelf CASE tool supporting UML.

1 Introduction

The research on agent-oriented software engineering starts from the possibility to model a software system at the agent level of abstraction [8]. This level of abstraction considers agents as atomic entities that communicate to implement the functionality of the system. This communication is supported by an agent communication language, such as FIPA ACL [5] or KQML [2], and by an ontology used to associate a meaning with content messages. Even if the agent level is the natural level of abstraction for describing a multi-agent system, the lack of accepted diagrammatic notations and design tools might prevent the multi-agent architect from exploiting its benefits. The research community of agent-oriented software engineering is tackling this problem investigating the possibility of extending UML to support the basic agent-oriented concepts such as agent, ontology and interaction protocol. This trend has led to a number of UML extensions [3, 9, 10, 13] that are not completely accepted because they are not supported by CASE tools yet.

This paper tackles the problem of defining a diagrammatic notation to support the design at the agent level exploiting an UML-based notation that can be managed by any off-the-shelf CASE tool. In particular, four agent-oriented diagrams are introduced taking advantage of stereotypes, the customisation means offered by UML. This allows providing an agent-oriented semantic to UML as a straightforward generalisation of its well-known object-oriented semantic and therefore the software architect should feel comfortable with it. The presented diagrams are meant to support the basic

agent-oriented abstractions, but they cannot be considered sufficient to support a complete agent-oriented model of the software life cycle. In particular, section 2 introduces *ontology diagram* as a means to provide the multi-agent architect with an UML-based notation to model ontologies. Section 3 tackles the problem of modelling the architecture of a multi-agent system introducing *architecture diagrams*. Section 4 completes the presented set of diagrams with *role diagrams* and *protocol diagrams*. These diagrams can be used jointly to model the interaction protocols that the multi-agent architect may use to co-ordinate agents. In order to show the possibilities and the features offered by the proposed agent-oriented diagrams, the agent-based service defined in the FIPA Audio/Video Entertainment and Broadcasting (*FIPA AVEB*) specification [4] is taken into account. This specification is sufficiently complex to represent a valid test case for the proposed diagrammatic notation and the comparison with the textual description found in FIPA documents should clarify the benefits of the presented approach.

2 Ontology Diagrams

The design of a multi-agent system at the agent level of abstraction requires defining a model of the world in which agents live. Agents exploit this model to reason about the world and to talk about it. This is fundamental in the design of a system at the agent level because this level of abstraction treats agents as atomic entities and therefore agents can be modelled properly only in terms of the messages they exchange.

The problem of describing the world to agents is traditionally solved providing them with an ontology that outlines a model of the world in terms of entities and relations between such entities [1, 6, 7, 12]. UML can be used to model ontologies exploiting ontology diagrams, i.e. class diagrams whose elements are associated with an ontology-oriented semantic. This semantic is straightforward as it is similar to the one employed in conceptual diagrams. An ontology diagram allows describing the entities belonging to an ontology in terms of classes. Such *entity classes* represent classes of entities comprised in an ontology, just like the classes in a class diagram represent classes of objects. Moreover, ontology diagrams allows the multi-agent architect defining the relations between the entities belonging to an ontology exploiting relations between entity classes. These are mapped into UML exploiting public relations between entity classes.

The agent level of abstraction does not deal with implementation details because these are normally tackled at the object level. This implies that entity classes are not allowed containing private and protected methods and attributes. Similarly, class diagrams allow associating a set of public methods to a class of objects to specify the messages objects may exchange to implement the functionality of the system. At the agent level of abstraction, this is not allowed because the actors communicating to implement the system are the agents and not the entity belonging to an ontology. Therefore, entity classes defined in an ontology diagram are allowed to be characterised only in terms of public attributes. These attributes are used to model the structure of the entities comprised in the ontology, just like they are used in conceptual dia-

grams. Figure 1 shows an ontology diagram describing a subset of the ontology defined in the FIPA AVEB specification. This diagram defines a class of entities called *FIPAAVDescription* characterised by a string called *title*. The stereotype *entity* is introduced to allow the multi-agent architect keeping the entities belonging to the ontology apart from the objects used at the object level. Moreover, this stereotype allows an agent-enabled CASE tool employing a different diagrammatic convention for object classes and entity classes, just like they normally do for actors and objects.

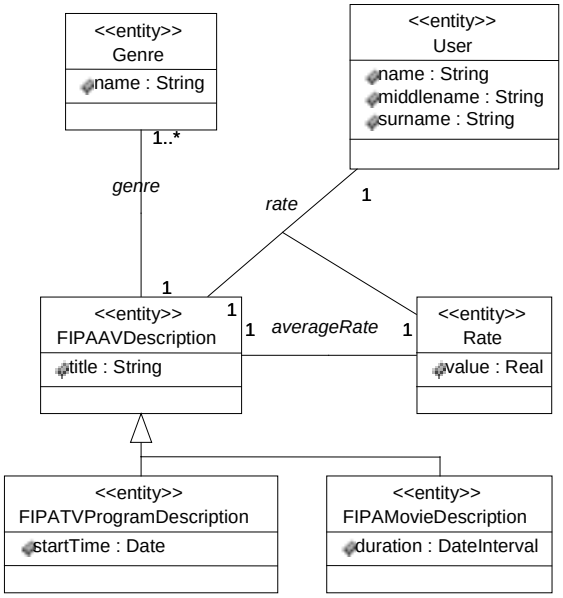

Figure 1. Part of the ontology diagram for the FIPA AVEB ontology.

Ontology-description languages allow modeling the relations between the entities comprised in an ontology. Ontology diagrams support this exploiting public relations between entity classes. For example, figure 1 shows that the classes *FIPATVProgramDescription* and *FIPAMovieDescription* extend the class *FIPAAVDescription*, i.e. these classes contain all attributes characterizing the class *FIPAAVDescription*.

The main purpose of ontology diagrams is to model the ontology followed by agents at the agent level of abstraction and the multi-agent architect may take this into account to produce useful ontology descriptions. In particular, ontology diagrams should contain the features needed to support the communication between agents. This can be accomplished treating relations between the entity classes as predicates defined over such classes. This approach allows the multi-agent architect exploiting an ontology diagram as a description of some of the elements, i.e. entity and predicates, that agents may use to create content messages. For example, figure 1 shows a relation called *averageRate* between an entity belonging to the class *User* and an entity belonging to the class *FIPAAVDescription*. This relation is chosen because the FIPA AVEB specification requires agents understanding messages like:

```
(inform
   :sender      guide-agent
   :receiver    control-agent
   :content
     (averageRate
       (FIPAMovieDescription :title “Batman”)
       (Rate :value 0.8)))
```

3 Architecture Diagrams

The research community of agent-oriented software engineering has not yet identified an accepted process to support the design of a multi-agent system. Nevertheless, all proposed processes emphasise the importance of modelling a multi-agent system in terms of an architecture that can be instantiated to create a configuration. Such architecture is composed of a set of *agent classes* and each class represents a class of agents with a precise set of responsibilities. UML can be used to model the configuration of a multi-agent system exploiting deployment diagrams. Besides, the modelling of its underlying architecture is not that easy. In fact, the architecture of a multi-agent system is generally specified by means of a textual description. This description contains a list of agent classes and a set of relations connecting them. Moreover, each class is associated with a set of responsibilities and with a set of messages that an agent belonging to that class is requested to understand. We propose to use architecture diagrams to model the architecture of a multi-agent system. These diagrams are class diagrams that contain agent classes and relations between such classes.

A complete model of an agent class requires describing the set of features used to characterise an agent belonging to that class. These features include:

1. the supported interaction protocols;
2. the accepted content messages, taking into account the ontology;
3. the semantic of each message.

Architecture diagrams support the latter two entries of this list, leaving the description of interaction protocols to protocol diagrams and role diagrams as described in the following section.

An agent class can contain only public methods corresponding to the actions that an agent belonging to that class can be requested to perform. These methods must be declared with no return values, i.e. they must be declared void. Moreover, the parameters passed to these methods must belong to entity classes defined in an ontology diagram. The list of actions associated with agent class does not include the performatives of the agent communication language because these are almost embedded in chosen the agent model. As an example, figure 2 shows the complete architecture diagram of the system defined in the FIPA AVEB specification. This diagram contains seven classes and just for one of them the FIPA AVEB specification provides a requirement regarding an action. Only agents belonging to the class *TunerWrapper* are requested to support an action, called *invoke*, to wrap concrete commands directed to a TV tuner. This action is understood only when supplied along with two parameters: a tuner, i.e. an entity belonging to the class *Tuner*, and a command expressed as a

String. Therefore, this diagram allows an agent belonging to the class *TunerWrapper* to accept messages like:

```
(request
   :sender     control-agent
   :receiver   tuner-wrapper-agent
   :content
     (action
       invoke
         (tuner   :name tuner1)
         (command "start")))
```

The actions that the multi-agent architect associates with the agent classes complete the elements provided by ontology diagrams to create valid content messages. An agent can understand all messages composed using the entities and predicates found in the ontology diagrams and can be requested to perform the actions specified in its class. This restricts agents understanding only messages based on first-order logic. While this is a limitation of the considered logic framework, it does not limit strongly the set of messages that agents may wish to exchange in a real-world multi-agent system. Therefore, the development of a more comprehensive logic framework is considered for a future work.

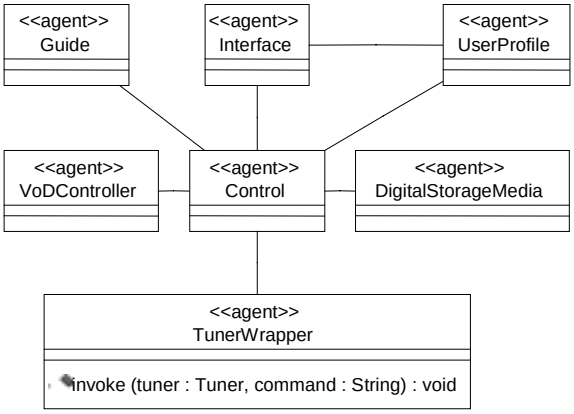

Figure 2. Architecture diagram of the system defined in the FIPA AVEB specification.

Architecture diagrams contain also relations between agent classes to allow the multi-agent architect modelling the relations between agents. These relations are mainly used to express simple acquaintance as it is common to promote flexibility in a multi-agent system avoiding the use of relations to spread the responsibility across agents. For example, figure 2 shows that the FIPA AVEB specification requires the control agent knowing all other agents in the multi-agent system, while the VoD controller agent knows only the control agent.

4 Protocol Diagrams and Role Diagrams

Interaction protocols are considered a valuable tool the multi-agent engineer may exploit in the design of a system at the agent level. They allow modelling the interactions between agents without taking into account the formal semantic of the chosen agent communication language. Moreover, they represent off-the-shelf solutions to a large number of problems.

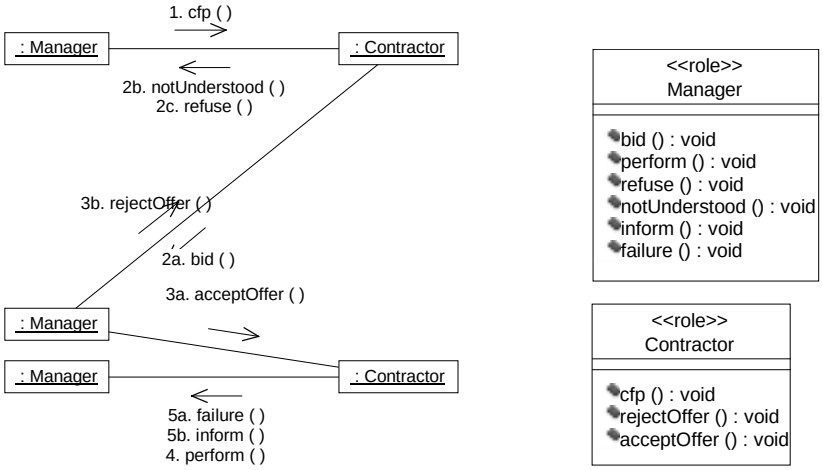

Figure 3. Protocol diagram and role diagram for the FIPA reactive contract net protocol.

Object protocols, the object-oriented counterpart of interaction protocols, are not considered a fundamental element in the design of a system at the object level. This is the main reason why UML does not provide diagrams to model object protocols. This has led to the assumption that UML should be extended to model interaction protocols at the agent level of abstraction. For example, AUML [10] defines a diagrammatic notation to modelling interaction protocols based, but not compatible, with sequence diagrams. A sequence diagram expresses a sequence of messages exchanged by a set of objects, while an AUML diagram models all sequences of messages that a set of agents may exchange in the scope of an interaction protocol. These diagrams are not generally adopted because they are not yet supported by off-the-shelf CASE tools. We propose to use protocol diagrams and role diagrams as an UML-based notation to model interaction protocols. These diagrams are based on collaboration diagrams and class diagrams respectively. A protocol diagram is a collaboration diagram that comprises only classes declared in a role diagram. These classes are labelled with the stereotype role to keep them apart from other kinds of classes, such as agent or entity classes. A role class models a role played by an agent in an interaction protocol. This role is meaningful only in the scope of the protocol in which it is defined and it is described in terms of the communicative acts that an agent playing that role must support. As an example, figure 3 shows the FIPA reactive contract net protocol defined in the FIPA AVEB specification.

5 Conclusions

Software engineering is founded on the possibility to model a system at different levels of abstraction. Agent-oriented software engineering introduces a new level, called the agent level, to allow the software architect describing a system in terms of interacting agents. At this level of abstraction, an agent is considered as an atomic entity that communicates with all other agents comprised in the multi-agent system exploiting an agent communication language. Besides, the multi-agent engineer may not exploit the benefits offered by this new level of abstraction for the lack of accepted notations and tools. This paper shows an UML-based notation that can be used to model a system at the agent level. This notation relies on the possibility to provide an agent-oriented semantic to the elements comprised in class and collaboration diagrams. In particular, four agent-oriented diagrams are introduced exploiting the customisation means, i.e. stereotypes, provided by UML to support domain-specific diagrams. These diagrams can be used to model the basic elements characterising the agent level of abstraction: the architecture of the multi-agent system, the interaction protocols supported by agents and the ontology followed by agents. Architecture diagrams allows modelling the architecture of the multi-agent system in terms of a set of agent classes connected through relations. Each class is characterised by the actions that an agent belonging to it can be requested to perform. The relations between classes may be used to express the network of acquaintance that an agent can gain in the architecture of the multi-agent system. Role diagrams and protocol diagrams are intended to support the multi-agent architect in the definition of interaction protocols. Role diagrams can be used to specify the roles that agents may play in a protocol, while protocol diagrams model an interaction protocol defined over a set of roles as the set of all possible conversations compatible with it. Ontology diagrams allows defining a model of the world composed of entities and relations. These relations can be used to specify the predicates that agents may use to communicate and therefore ontology diagrams can be used to model the content messages that agents are allowed to use to communicate.

Acknowledgement

This work is partially supported by the EU IST Project LEAP (IST-1999-10211).

References

1 S. Cranefield and M. Pruvis, *"UML as an Ontology Modelling Language"*, in Proceedings of the Workshop on Intelligent Information Integration, 1999.
2 T. Finin, Y. Labrou and J. Mayfield, *"KQML as an Agent Communication Language"*, in J. M. Bradshaw (Ed.) Software Agents, MIT Press, 1997.

3 FIPA Technical Committee C, *"Extending UML for the Specification of Agent Interaction Protocols"*, response to the OMG Analysis and Design Task Force UML RTF 2.0 Request for Information, 1999.
4 FIPA, *"FIPA '97 Specification Part 6: Audio/Video Entertainment and Broadcasting"*, available at http://www.fipa.org.
5 FIPA, *"FIPA '99 Specification Part 2: Agent Communication Language"*, available at http://www.fipa.org.
6 M. R. Genesereth and R. E. Fikes, *"Knowledge Interchange Format - Version 3 - Reference Manual"*, technical report Logic-92-1, Stanford University, 1992.
7 M. R. Genesereth, N. Singh and M. Syed, *"A Distributed and Anonymous Knowledge Sharing Approach to Software Interoperation"*, International Journal of Cooperative Information Systems 4(4):339-367, 1995.
8 C. A. Iglesias, M. Garijo and J. C. A. González, *"Survey of Agent-Oriented Methodologies"*, in Proceedings of the Workshop on Agent Theories, Architectures and Languages, 1998.
9 J. Odell and C. Bock, *"Suggested UML Extensions for Agents"*, response to the OMG Analysis and Design Task Force UML RTF 2.0 Request for Information, 1999.
10 J. Odell, H. Van Dyke Parunak and B. Bauer, *"Representing Agent Interaction Protocols in UML"*, in Proceedings of Agents 2000, 2000.
11 OMG, *"The Common Object Request Broker Architecture and Specification"*, available at http://www.omg.org.
12 P. F. Patel-Schneider and B. Swartout, *"Description-Logic Knowledge Representation System Specification"*, DARPA KSE technical report, 1993.
13 J. Treur, *"Methodologies and Software Engineering for Agent Systems"*, available at http://www.cs.vu.nl/~treur.