Integrating Ontology Support within AgentService

Christian Vecchiola, Alberto Grosso, Antonio Boccalatte
DIST – Department of Communications Computer and System Sciences
University of Genova
{christian, agrosso, nino}@dist.unige.it

Abstract

This paper describes the software infrastructure introduced within the AgentService framework in order to provide support for ontology design, development, and management. Ontology enriches and normalizes the interaction among agents by establishing a domain and a set of relation among objects populating that domain. A good support for ontology definitely adds value to the design and the implementation of software agents: software engineers can take advantages of the services offered by the framework to produce new ontologies and rely on them to quickly define interaction protocols which are automatically translated into state machines used by software agents.

1. Introduction

In computer science the word ontology refers to "a data model that represents a domain and is used to reason about the objects in that domain and the relations between them" [1]. Software ontologies are used in different fields of computer science such as information architecture, semantic web, and knowledge representation. In particular, the adoption of software ontologies for knowledge representation is very attractive: ontologies contribute to provide a structure, a collection of well identified concepts along with their properties, and relations among them to a given knowledge base. For these reasons, they are very useful for software agents that base their activity mainly on the interaction with peers and on reasoning about the environment. Ontologies provide a structured and efficient way to perform these tasks.

The translation of software ontology into a collection of software artifacts representing it delivers to MAS engineers a high level of abstraction helping them in defining the interactions among agents. From a practical point of view, a given software ontology establishes the content of messages exchanged among agents and provides facilities to validate them. Moreover, ontologies are a good starting point for defining interaction protocols which are the most common way to define a structured dialogue among two entities. Hence, a good support for ontology design, development, and management, definitely gives an added value to agent programming frameworks since it simplifies and empowers the activity of MAS engineers.

This paper presents the collection of software abstractions and tools integrated into AgentService [2] which provides the framework with ontology design, development, and management (hereafter ontology service). The ontology service has been designed by following the specifications provided by FIPA [3]. In the next sections we will give a brief description of AgentService and the agent model it proposes (Section 2), then we will mostly concentrate on the entire process of defining, implementing an ontology and using it to support interaction protocol design and implementation (Section 3 and 4). Conclusions will follow.

2. AgentService

In this section we will give a brief overview of the AgentService framework by pointing out only those aspects which are relevant to understand how the ontology service is integrated into the framework. Basically, we will describe the components of the framework and we will present the agent model. For a more detailed introduction please see [2].

2.1. The Framework

AgentService is a framework to implement distributed multi-agent systems. It provides support for agent design and implementation, multi-agent system implementation management and monitoring. The components which constitute the framework are the following:

- a flexible agent model through which different agent architectures can be implemented;
- a library which defines the core of the system and the basic services of the framework;
- a software environment that hosts multi-agent systems and controls their life-cycle;

- a set of programming language extensions simplifying the implementation of software agents;
- a collection of tools supporting users in designing and implementing multi-agent systems;
- complete support for ontology definition and development;
- automatic code generation for interaction protocols with ontology integration;
- a software infrastructure allowing agents to migrate among different instances of the AgentService platform;
- a set of support programs through which users can maintain and monitor multi-agent systems.

The core of the framework relies on the Common Language Infrastructure (hereafter CLI) [4] and makes the framework portable over different implementations of this specification like Mono, Rotor, and .NET. The key features of the framework are the agent platform which is a modular hosting environment for software agents and the agent model which will be investigated in the next paragraph.

2.2. The Agent Model

The framework defines a software agent an autonomous *software entity whose activity is constituted by a set of concurrent tasks and whose state is defined by a set of shared objects*. Concurrent tasks are referred as *behaviour objects* while the term *knowledge object* is used to identify the components of the agent state.

Behaviour objects encapsulates all the computational activity of a given software agent while knowledge objects define the elements composing its knowledge base. The formers can be considered as simple little programs which have their execution stack and can communicate each other by using the shared knowledge objects. Behaviour objects can access the runtime services of the agent platform and query the FIPA management agents (AMS, MTS, DF, and Ontology Agent) in order to obtain information about the environment, the community of agents and the services they offer; for example they can query the Ontology Agent in order to know which ontologies are registered in the platform and which agents are able to understand messages belonging to a given ontology.

Knowledge objects are data structures containing items which are exposed as properties. They resemble a C struct or a Pascal record, but are designed with a built support for persistence and concurrent multiple accesses. Knowledge objects define the knowledge base of a given software agent and the collection of their properties along with the execution state of each agent define the state of an agent instance.

Agents can interpret roles into a given communication protocol and they can publish this service through the DF which makes this information available to the entire community of agents. Interpreting a role makes the agent able to participate into the communication protocol which defines that role. This feature is implemented by providing the agent with a behaviour object which automatically executes the state machine defining the role. This issue will be further detailed in the next section.

3. Ontology Support in AgentService

AgentService provides a complete support to ontology design, implementation, and management. These three functionalities are collectively referred as the *ontology service*. The ontology service is based on the following framework components:

- a set of classes representing the object model defining all the elements required to represent an ontology (classes, concepts, instances, attributes, constraints, validation, etc);
- a set of tools that can be used to automatically generate the specific classes for a given ontology by starting from its visual or textual representation;
- an Ontology Agent (OA) which maintains the knowledge about all the ontologies registered in the hosting agent platform and about the agent which are able to communicate by using the concepts defined into a given ontology;
- FIPA SL0 [5] ACL message support.

As we can notice, from the previous list the framework does not directly provide any facility to visually design software ontologies. We decided to rely on a very well know and established tool that is Protégé [6] a software projects maintained by the KSI lab the Stanford University. Protégé, when equipped with Jambalaya [7], provides all the required features to quickly design a given ontology. In the following we will briefly illustrate the object model designed to support ontology definitions in AgentService, the role of the Ontology Agent, and the ontology development process.

3.1. Ontology Object Model

The design and the implementation of the object model defining the ontology reflects the specifications outlined in the corresponding FIPA standards [3] and has been inspired by the type system designed in JADE [8] to support ontologies. The object model defined within AgentService defines a meta-ontology which contains all the concepts and the elements which are required to compose user defined ontologies. The meta-ontology

defines the following entities: *predicate*, *term*, *concept*, *query*, *action*, *variable*, *primitive*, and *aggregate*. Figure 1 describes how these elements are connected each other.

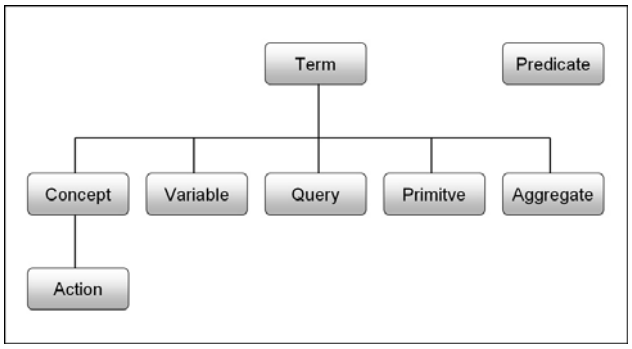

Figure 1. Ontology elements hierarchy

The elements depicted in figure 1 define the domain in which every communication based on a given ontology takes place. User defined ontologies will provide specific instances of these elements and MAS engineers will have to specialize the abstract classes representing the entities defined by the meta-ontology: the new classes will represents the concepts, the queries, the actions, the terms, etc. which are pertaining to the specific problem domain. Ontologies are also described using *schemas* which are generic objects used to describe any ontology element and provides all the information to represent a specific ontology without using ontologies specific classes. Ontology schemas are used internally by the framework in order to maintain a catalog of all the ontologies registered, while ontologies specific classes constitute the API used by software agents to hard-code the communication based on a specific ontology. Agents which can dynamically learn to use a given ontology by exploiting the services of the *OntologyDescriptor* class which automatically extracts all the useful information about a given ontology. It retrieves all the schemas defining the specific entities of the ontology and provides also other useful information such as assembly location, file versioning and type names. The implemented solution is very flexible since it provides an efficient way of using ontologies when they are statically identified, while, at the same time, provides facilities to dynamically reflect[1] ontologies.

AgentService also provides support for SL0 which is a minimal subset of the FIPA ACL message specification. The framework defines a set of classes able to represent all the elements of SL0 plus the = (equals) predicate. In order to communicate with a given ontology agents exchange messages which contains SL0 objects and a specific message content has been designed (the *ACLMessageBody* class) in order to transport SL0 statements.

The joint use of the ontology API and of SL0 as a vehicle allows agents to easily communicate each other.

3.2. The Ontology Agent

In order to be compliant with the specification provided by FIPA we have introduced the Ontology Agent which is responsible of maintaining the catalog of all the ontologies registered with the system and of providing useful information to software agents. The entire list of task that should be performed by the ontology agent is the following:

- ontology discovery and publishing;
- ontology maintenance;
- ontology mapping and translation;
- shared ontology discovery.

The ontology agent provided with the framework implements only the two features of the previous list which are also the most important. We think that the ontology mapping service is a very difficult task to implement and requires some sort of inductive knowledge in order to detect similarities among different knowledge representations.

In order to be available to the community of agents the Ontology Agent registers its service to the DF. Since we have implemented a reduced set of task of the ontology agent we decided to embed these functionalities directly into the Directory Facilitator, by adding a specific behaviour which performs these tasks. The community of agents asks to the DF which agent provides the ontology service and the directory facilitator returns its own agent identifier. Hence, the implementation of this feature is completely transparent to the community of agents which only expect to obtain the address of the Ontology Agent in order to query it.

3.3. Ontology Development Process

In order to make users feel at ease and increase their productivity we adopt, as written above, Protégé in conjunction with Jambalaya for defining AgentService ontologies. Figure 2 describes the entire process that by starting from the Protégé editor generates the assembly containing the type definitions for the ontology which are required by the AgentService framework.

Projects designed in Protégé can be exported into the XML format and a tool provided with AgentService automatically generates the corresponding object model, writes the source code and compiles it into an assembly

[1] The verb reflect is used in the sense of *type reflection* which identifies a well know set of operation aimed to extract information about the class type of a given object.

which can be easily deployed into the agent platform or used by the protocol designer. After the compilation process takes place MAS engineers are provided with the entire object model describing the ontology they designed with Protégé. The assembly can be included into a generic software project and used as a library, directly deployed into the agent platform, or, as showed in figure 2, used within the protocol designer.

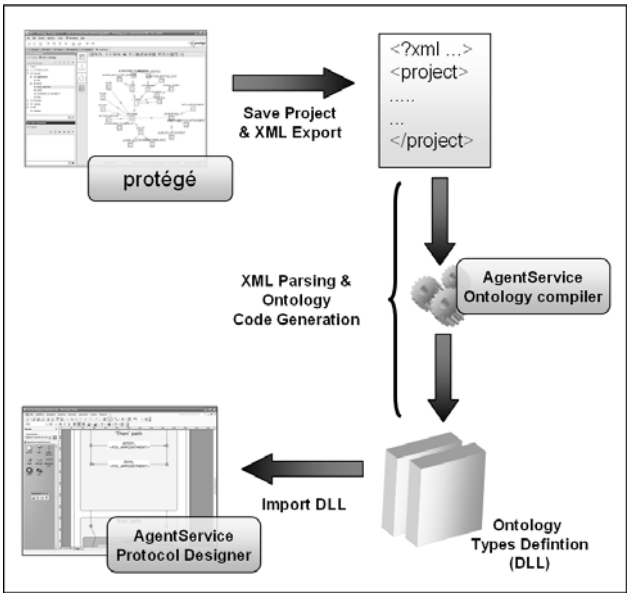

Figure 2. Ontology code generation

4. Protocol Design and Implementation

AgentService provide facilities to design, to implement, and to integrate interaction protocols into multi-agent systems. Interaction protocols rely on the ability of agents of communicate each other by using the services offered by the MTS and the messaging subsystem. The development of a protocol for AgentService can be enhanced by using ontologies which give a sound meaning and a well defined structure to the messages exchanged during the interaction. AgentService relies on the Microsoft Visio visual modeling environment and provides a plug-in which allows users to define interaction protocols by following the AUML standard [9].

In particular by using protocol designer developers can:

- design protocol steps for each agent role involved;
- adopt previously defined ontology for message content;
- export the protocol in a format usable by AgentService;
- design agents interpreting the roles through dedicated behaviours.

The plug-in introduces a new stencil in the Visio environment which contains all the elements to visually compose the protocol, and a toolbar which allows to automatically generate the code implementing the state machines for the protocols. Such state machines can be embedded into specific behaviour objects which are then able to participate in the interaction protocol.

The next paragraphs analyze more in details how the tool works, in particular some extensions of the AUML basic elements are introduced and the code generation process is explained.

4.1 Designing AUML elements

Agent UML proposes extensions to UML and idioms within UML in particular for sequence e collaboration diagrams. Sequence diagrams seem to be the best way to design and represent agent in order to capture inter-agent dynamics [10]. Two fundamental parts constitute the diagram model a frame, which delimits the sequence diagram, and the message flow between roles through a set of lifelines and messages [11]. Hence the frame element contains lifelines, sets of messages, and AUML operators commonly called *combined fragments*. In addition to the basic elements proposed by AUML specification (lifeline, message, alternative, optional, break, loop), we introduce some new features in order to make the model effective from the AgentService point of view.

Since the definition of the protocol is designed in order to generate running code for AgentService platform, the tool provides features for inserting lines of .NET code for each role. In particular designers can write code in order to manage exceptions during protocol execution or reply to an error message received from a peer.

Messages are obviously the core of protocol interactions; the tool provides two kinds of messages: ontological messages and native messages. The designer is integrated with the AgentService ontology system; in order to adopt an ontology it is only necessary to indicate the assembly containing it. Hence if a user wants to use an ontology within a protocol, he has to select an SL0 operator and then choose the ontological elements contained within the previously loaded ontology. In the case the user does not adopt an ontology, he has to define the fields composing the body of the message, specifying the name and the type of the message element (the message content of AgentService is strongly typed).

The tool allows developer to design *peer-to-peer* or *client-server* communications defining the cardinality of the agent roles involved in a protocol. AgentService messages are asynchronous; in client-server protocol the

reception by the server is time-outed and the server manages client interaction by adopting a round-robin approach.

The AUML tool for AgentService allows users to define guard conditions (for alternative and optional statements) as Boolean expressions which very often involve elements contained within messages previously received. Hence the guard condition is not a simple label but it is a fundamental element involved in the generation of the code modeling the protocol.

Finally the AUML Loop operator is extended in order to be able to clearly indicate the agent role that manages the loop.

4.2 Code generation and execution of interaction protocols

Starting from the model representing the interaction protocol it is import to be able to generate agents playing roles within the protocol and then execute them. The designer only defines the structure of the protocol and creates classes targeting the AgentService object model which represent the state machine executing the interaction protocol. It is up to the agent playing a specific role to complete the state machine generated by the tool with all the information it needs.

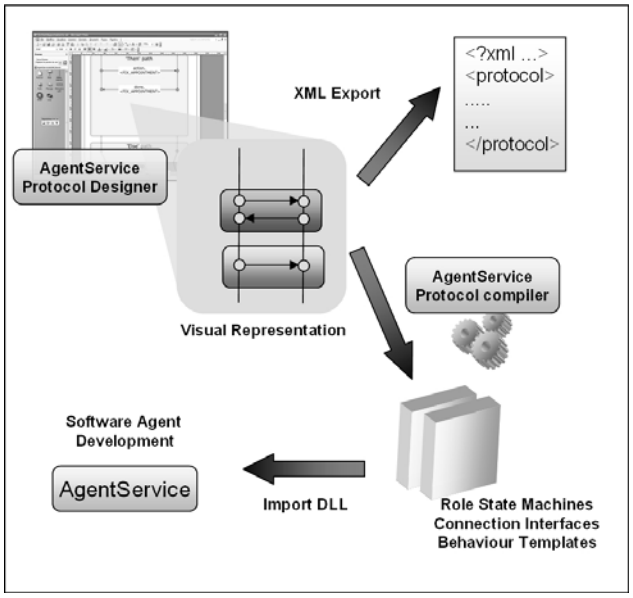

Figure 3. Protocol development process

Figure 3 describes the protocol development process: interaction protocols can either be exported into XML format or into assemblies which can be directly used to program software agents. Both of the two representations contain the same information: the first is mainly used to maintain a textual representation of the protocol and to export it to third party applications, while the second is the most important since it actually allows the use of protocols inside the framework. Given an XML or a visual representation of the interaction protocol, the protocol compiler generates an assembly containing the following entities:

- the definition of the protocol object model;
- the definition of state machines implementing the roles defined in the interaction protocol;
- the definition of the interfaces types used by the state machines to customize the execution flow of the protocol.

Software agents, in order to participate into a protocol, have to interpret one of the roles defined in the protocol. This role is interpreted by adding a behaviour object which executes the state machine defining the role; such behaviour object has to implement the interface required by the state machine and through the methods exposed by this interface interacts and controls, when possible, the execution of the state machine. The customization level provided by interface is required, for example, in order to let the agent choose among different alternatives and then drive the execution of the protocol.

The state machines automatically handle all the exceptions that can occur while executing the protocol (i.e. wrong or malformed messages, etc.) and the behaviour objects running the state machine can be informed about these exceptions through specific events.

4.2 Related works

There are some interesting works based on Agent UML involving the definition of design tools. Ehrler and Cranefield propose a Plug-in for Agent UML Linking (PAUL) [12] based on the FIPA-compliant agent platform Opal [13] and the Eclipse Modeling Framework. In particular this tool uses the UML Object Constraint Language (OCL) [14] in order to define the input and output of the operations an agent can perform within a protocol. Our design tool does not require the definition of I/O data for protocol operations; agents playing a protocol role have access to all the content of the messages received during the interaction and of course can consult their knowledge base.

Winikoff [15] precisely defines the syntax of a subset of AUML by using a textual notation; he provides a tool for designing diagrams in order to support the notation. This tool cannot generate code for the models that have to be implemented manually.

Viper [16] is a graphical editor based on the earlier version of AUML that can generate code for AgentFactory [17]. The tool, in the implementation phase, involves users for populating the protocol with customized agent code, which together with code

automatically generated to reflect the protocol semantics is compiled into useable agent designs. Hence the tool generates a skeleton of code implementing the protocol and users have to complete and compile it. Instead the AgentService protocol designer generates an assembly containing the classes representing the state machine of the protocol and a programming interface that users can implement in order to customize agent operations. Viper is not provided with an ontology system.

5. Conclusions

In this paper we presented the software infrastructure introduced into the AgentService framework in order to support ontology design, implementation, and management. Software ontologies are a high level abstraction which is very useful for identifying the concepts of a problem domain, to define their relation, and to reason about them. Ontologies give an added value to the interaction among software agents since they provide facilities to define a communication and to validate messages.

The support provided by AgentService covers, either directly or not, all the activities previously cited: AgentService relies on Protégé in order to define a new ontology and translates the representation provided by Protégé into a collection of classes fitting the object model defined into the framework. These classes can be easily used to define a conversation among software agents: messages can be verified against a specific ontology and eventually discarded if spurious. By querying the ontology agent we can dynamically inspect the ontology catalog maintained in every multi-agent system and extract useful information about their structure, such information can be easily used in order to start a communication with agents which know that ontology.

Software ontologies can also be useful to define structured conversations among agents since they completely define the content of the exchanged messages. AgentService provides a useful tool to visually define interaction protocol integrated with the ontology service: it is then possible to start from the definition of the problem domain with Protégé, translate that representation into a software ontology, build an interaction protocol based on those concepts, and produce a state machine which can be directly used by software agents to interpret roles. All this process can be performed without writing a line of code thanks to the supports provide by the framework.

Nonetheless, the ontology service can be improved: now AgentService provides support only for the SL0 subset of the FIPA ACL specification while a complete implementation of the SL2 specification is still to come. Moreover, a more complete service provided by the ontology agent has to be implemented in order to be completely compliant with the FIPA specifications.

6. References

[1] Wikipedia, definition of software ontology, available at http://en.wikipedia.org/wiki/Ontology (computer sci-ence).

[2] C. Vecchiola, A. Grosso, A.Gozzi, and A. Boccalatte, "AgentService", *Proceedings of the 16th International Conference on Software Enginnering and Knowledge Engineering (SEKE04)*, Banff, Alberta Canada, KSI Publisher, 2004.

[3] FIPA, "FIPA Ontology Service Specification", 2001, [Online document], Availabe at HTTP:http://www.fipa.org/specs/fipa00086/XC00086D.pdf.

[4] Standard ISO/IEC 23271:2003: Common Language Infrastructure, March 28, 2003, ISO.

[5] FIPA, "FIPA SL Content Language Specification", [Online document] available at HTTP: http://www.fipa.org/specs/fipa00008/SC00008I.html.

[6] J. Gennari, M. A. Musen, R. W. Fergerson, W. E. Grosso, M. Crubezy, H. Eriksson, N. F. Noy, and S. W. Tu, "The Evolution of Protégé: An Environment for Knowledge-Based Systems Development", Stanford Knowledge System Lab, 2002.

[7] M. Storey, M. Musen, J. Silva, C. Best, N. Ernst, R. Fergerson, and N. Noy, "Jambalaya: Interactive visualization to enhance ontology authoring and knowledge acquisition in Protégé", *Workshop on Interactive Tools for Knowledge Capture*, Victoria, BC, Canada, 2001.

[8] G. Caire, "Jade Tutorial – Application-defined Content Languages and Ontologies", 2002.

[9] J. Odell, H. V.D. Parunak, and B. Bauer, "Extending UML for agents", *Proceedings of Agent-Oriented Information Systems Workshop (AOIS-00)*, 2000.

[10] J. Odell, H. V.D. Parunak, and B. Bauer, "Representing agent interaction protocols in uml", *Paolo Ciancarini and Michael J. Wooldridge Editors, Agent-Oriented Software Engineering*, No. 1957 in LNCS, pp. 121–140, Springer Verlag, 2000.

[11] FIPA. "FIPA Interaction Protocol Library Specification", 2001. [Online document], Availabe at HTTP: http://www.fipa.org/specs/fipa00025/XC00025E.pdf.

[12] L. Ehrler and S. Cranefield, "Executing agent UML diagrams", *Proceedings of the third international conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pp. 906–913, 2004.

[13] M. Purvis, S. Cranefield, M. Nowostawski, and D. Carter, "Opal: A Multi-Level Infrastructure for Agent-Oriented Software Development", *Information Science Discussion Paper Series*, No. 2002.

[14] J. B. Warmer and A. G. Kleppe, "The Object Constraint Language: Getting your models ready for MDA", *Addison-Wesley*, 2nd edition, 2003.

[15] M. Winikoff, "Towards Making Agent UML Practical: A Textual Notation and a Tool", *First international workshop on Integration of Software Engineering and Agent Technology (ISEAT 2005)*, Melbourne, Australia, 2005.

[16] C. Rooney, R. Collier, and G. M. P. O'Hare, "VIPER: Visual Protocol EditoR", *6th International Conference on Coordination Languages and Models (COORDINATION 2004)*, 2004.

[17] R. Collier, "Agent Factory: A Framework for the Engineering of Agent-Oriented Applications", *Ph.D. Thesis*, Department of Computer Science, University College Dublin, Ireland, 2001.