Argonaut: Integrating Jason and Jena for context aware computing based on OWL ontologies

Douglas Michaelsen da Silva[1], Renata Vieira[1]

[1] Universidade do Vale do Rio dos Sinos
Av. Unisinos, 950 - CEP 93.022-000 São Leopoldo - RS - Brasil
michaelsen@gmail.com, renatav@unisinos.br

Abstract. In this paper, we present the integration of the agent-oriented programming framework Jason and the semantic web framework Jena to support ontology-based context aware computing. These technologies together allow for the development of context aware multi-agent systems base on ontologies that describe context.

Keywords: Semantic Web, Ontologies, Agents, Context aware computing.

1 Introduction

The Semantic Web project and the related development of applications that make use of knowledge resources are attracting much of current research interest. The Semantic Web proposed technologies are also proving adequate as a basis for other important areas of Computer Science such as Ubiquitous Computing [3]. These new computing technologies are changing the way users interact with applications. These changes are due to the fact that users, devices and applications are given mobility. In this scenario, context awareness is a relevant requirement for applications. The computational representation of context is thus a growing field of research and technology development. Semantic Web technologies currently available, such as description logic based ontologies and intelligent agents are promising solutions for context representation and manipulation. These technologies allow pro-active contextual help and guidance for mobile users and applications.

Although applications for the Semantic Web are already being proposed, often based on agents paradigm, most of such efforts does not consider proper agent-oriented programming languages. Besides, web ontology languages and agent oriented programming languages have both been developed independently from each other. On the other hand, the integration of such agent oriented programming languages, such as AgentSpeak [1], with automatic reasoning over ontologies can have a major impact on the development of agents and multi-agent systems that can operate in a SemanticWeb context. In fact, the theoretical aspects of such integration have been already proposed [5]. However, the practical integration of such technologies for developing real applications is still a challenge.

In this work, as a first approach to explore the integration of these technologies in a practical way, for the development of typical mobile computing applications, we show an AgentSpeak/Jason[1] prototype in which BDI agents access an OWL ontology through the Jena framework[2]. The prototype implements agents that help users to find out about locally situated services.

The paper is organized as follows: Section 2 presents an ontology that describes contextual information; Section 3 presents an overview of the agent programming language AgentSpeak; Section 4 presents the integration of Jason and Jena for a context aware application prototype; Section 5 concludes the paper.

2 Argonaut Ontology

Among the key components of the Semantic Web are *domain ontologies* [7]. They are the proposed model for knowledge resources, underlying specific web languages. Ontologies are therefore the component responsible for the specification of the domain knowledge. As they can be expressed logically, they allow for reasoning in the specified domain. Indeed, several ontologies are being proposed for the development of a large variety of applications [2, 3].

OWL is a language developed for representing ontology information on the semantic web. OWL is based on descriptions logics which are appropriate for ontological reasoning. OWL can be used to describe concepts and their relationships as well as specific properties and restrictions through logical axioms. According to different underlying logics, there are three versions of OWL: OWL Full, OWL DL and OWL Lite.

OWL ontologies have been developed for ubiquitous and pervasive applications; the SOUPA ontology is an example [2]. It was designed to support mobile applications, its vocabulary is derived from other existing ontologies, some examples are: FOAF, an ontology for personal relationship information, people and their basic data such as address, phone number, e-mail, etc; DAML-Time, an ontology for common knowledge about time and temporal events; Spatial ontologies (such as OpenCYC and RCC) for spatial concepts and reasoning about localization.

Since ontologies are to be shared and reused, we developed a small ontology by adapting some of the main concepts and relations of SOUPA. We created instances corresponding to a university environment. The main concepts adapted were *Person*, *Geographic Space* and their subclasses. New concepts were created to accommodate the application we had in mind. The new concepts are service and distance. We used Protégé [4] and its OWL plugin to build the ontology that is the basis of our application. The classes *Person* and *Service* describe, respectively, the users of the application and the services provided in different regions of the campus. The relation "at" (for is situated at), shown in Figure 1, holds for persons and services with geographical spaces.

[1] http://jason.sourceforge.net/
[2] http://jena.sourceforge.net/

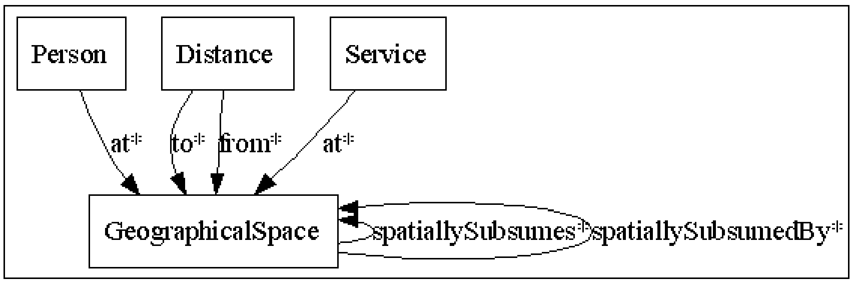

Fig. 1. Relationships among concepts.

The concept *GeographicalSpace* can be specialized as *GeographicalRegion*, *FixedStructure* (used for buildings) and *SpaceInAFixedStructure* (used for rooms). A geographical space may be spatially subsumed by another one. For example, a fixed structure can be spatially subsumed by a geographical region and can spatially subsume a space in fixed structure. For example, buildings can be situated in a campus and they can contain rooms. The concept *Distance* relates to geographical spaces through the relations "from" and "to". All instances of this concept represent a distance between two spaces. In a specific situation we could have the user Maria, who is an instance of *Person*. She is located at LabOne which in turn is a *SpaceInAFixedStructure*. The instance BuildA spatially subsumes the instance LabOne and is spatially subsumed by instance CentreX. In this situation agents that perceive the presence of Maria in LabOne can infer that Maria is at BuildA in CentreX.

3 AgentSpeak

As ontologies, agents are also considered a fundamental component of the semantic web. Agents are responsible for helping the users in their service requests. They can make use of the available knowledge; autonomously interact with other agents, so as to act on the user's interest. Of course, on the view of the Semantic Web agents can only achieve these requirements by sharing domain ontologies.

Here we consider the agent oriented programming language AgentSpeak. Jason is the interpreter for AgentSpeak, which is available Open Source under GNU LGPL at http://jason.sourceforge.net. It implements the operational semantics of AgentSpeak given in [8]. AgentSpeak provides an elegant abstract framework for programming agents. An AgentSpeak agent (or program) is defined by a set of beliefs, which is a set of ground (first-order) atomic formulas, and a set of plans which form its plan library. AgentSpeak distinguishes two types of goals: achievement and test goals. Achievement goals are formed by an atomic formula prefixed with the '!' operator, while test goals are prefixed with the '?' operator. An achievement goal states that the agent wants to achieve a state of the world (and it will look for a stated plan in his plan library for that). A test goal states that the agent wants to test whether the associated atomic formula is (or can be unified with) its beliefs. Being a reactive planning system, the events it reacts to are related either to changes in its beliefs due to perception of the environment, or to changes in the agent's goals that originate from the execution of plans triggered by previous events. A triggering event can trigger the execution of a particular plan. Plans are written by the programmer so that they are triggered by the addition ('+') or deletion ('-') of beliefs or goals (the "mental attitudes" of AgentSpeak agents). These elements are exemplified in Figure 2.

Consider a scenario where a student or academic visitor is walking around the university campus. The student may be notified about locally available services, or scheduled invited talks, according to the user's preferences. In the example (Figure 2) we show some AgentSpeak plans for this scenario.

```
+lecture(A,V,T) : interested_in(U,A)
  !inform(U,A,V,T)

+!inform(U,A,V,T) : ¬busy(U,T)
  show(U,A,V,T)…
```

Fig. 2. Examples of AgentSpeak plans.

The first plan tells us that, when a lecture A is announced at venue V and time T (so that, from the perception of the context, a belief lecture(A,V,T) is *added* to the belief base of the agent), then if a user U is interested in A, it will have the new goal of inform interested users for that lecture. The second plan tells us that whenever this agent adopts the goal of informing users about lectures, if it is the case that the user is not busy at T, according to his agenda, then it can proceed to execute that plan consisting of performing the basic action show(U,A,V,T) (assuming that it is an atomic action that the agent can perform). This brief introduction will help to guide the reading of the next section, in which we present our prototype. More details about AgentSpeak can be found in [1].

4. The Argonaut prototype

Argonaut is a multi-agent system that integrates Jason and Jena in order to allow agents to interact on the basis of contextual knowledge represented in an OWL ontology. Jason provides the means for the specification of the environment in which the agents actuate. In the specified environment, agents perceive the user and user requests, they communicate with each other to provide the user information relative to their distance to required services. Through some defined internal actions the agents consult an OWL ontology that contains contextual information. The interaction with the OWL ontology is done through the Jena framework.

Our scenario is such that the user is located somewhere in the university campus and he intends to know about the available services nearby. He also intends to know the distance from his location to some required service (library, food court, computers lab, shops, etc). The mobile device is perceived by a server and the local interface agent communicates with him offering the locally available services. When the local interface agent perceives the arrival of the user, it greets him and then shows him the locally available services. The user selects one service, the interface agent then asks about the location of the selected service to the location agent. The location agent gets the service and the user location and queries the distance between the user and the service, returning it to the interface agent. The interface agent shows to the user his distance from the required service.

In our prototype the presence of the user is simulated, the service selection is done through the user interface and they are external events that occur in the environment. The interface agent perceives the user selection and communicates with the location agent, which is responsible for consulting the context ontology. Events and actions are implemented in Jason. External events are the user arrival, which is perceived by the environment, and the selection of one available service, made by the user. Internal actions do not modify the environment, in the Argonaut they are responsible for consulting the OWL ontology through the Jena framework. Queries to the ontology return contextual information which is added in the agents' belief base.

The system overview [6] is illustrated in Figure 3. Events are represented by stars and actions by arrows. The two actions named with *jena* corresponds to internal actions of the agents for querying the contextual OWL ontology. The result for such queries is contextualized information that is stored in the agents' belief bases.

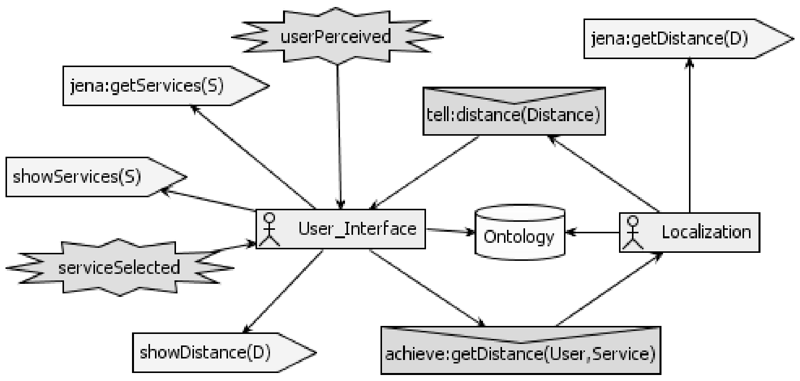

Fig. 3. Argonaut system overview.

The agents are named *localization* and *user_interface*, there is a local interface for each University Institute. Each Institutue, or region of the University, would delegate an agent to serve as its local service interface. In this way, each locally perceived user is tied to a local user interface agent. The prototype follows a MVC (Model View Controller) architectural pattern. The model-view-controller separates data access and business logic from data presentation and user interaction, by introducing an intermediate component: the controller. The environment where the agents actuate corresponds to the Controller module, which is also an observer to the other modules. It is necessary because each event that happens in the View or in the Model need to be handled by the Jason environment. Therefore, the Model module implements the consults to the ontology, which are made through RDQL/Jena, and the View module is responsible for the interface of the prototype, implementing some functionality to handle graphical configurations.

These two modules implement the observable class that notifies the environment when something happens. In this case, when there are relevant plans to deal with events they are activated. An example is when a service is selected by the user, the environment is notified, the agent perceives this change and the corresponding plan is activated, as explained below.

In Jason, the environment is responsible for providing events (perceptions) to the agents, in our prototype these events are the presence of the user and the selection for a service made by the user through the graphical interface. The available services are described in the ontology and they are presented to the user by the interface agent.

Next we describe the plans of the *user_interface* agent (Figure 4). A belief about the presence of the user is added to the belief base of the agent. This triggers the plan for the arrival of a new user, a plan to show the available services. The second plan is triggered when the user selects a service. First, an action to exhibit the selected service is executed. A test goal identifies who is the user perceived, then an achieve message is sent to the localization agent, so that an appropriate plan of the localization agent is triggered.

```
+userWasPerceived(User):
       <- ont.GetServices(Centre,S);
       showServices(S).


+selected(Service)  :  true
       <- .print(Service);
       ?userWasPerceived(User);
       .send(localization, achieve,
       getDistance(User, Service)).


+distance(D) : true <- showDistance(D).
```

Fig. 4. User_interface agent.

The localization agent (Figure 5) executes the plan with two internal actions: an action that consults the ontology and queries the distance between the user and the agent; and a *.send* action to return the resulting distance to the agent that sent the achieve message. This *tell* message will cause a belief addition in the belief base of the user_interface agent, triggering the plan for showing the retrieved distance to the user.

```
+!getDistance(User, Service)[source(SAg)] : true
      <-
      ont.GetDistance(User, Service, Distance);
      .send(SAg, tell, distance(Distance)).
```

Fig. 5. Localization agent.

In our prototype, the locally available services and distance between locations are described in the OWL ontology. Ontology queries are done with Jena, an open source

Java framework for building Semantic Web applications. It has an API that aims to provide a consistent programming interface to the semantic web application developer. It provides an environment for querying ontologies and includes a rule-based inference engine. Consults using RDQL[3] (RDF *Data Query Language*) are done using this model. RDQL consists in a graph of triples. Each triple contains variables which are instantiated with the corresponding required values.

The localization agent executes the action *GetDistance* that returns the distance between the user and the requested service. For example, Maria is at LabOne, which is subsumed by BuildA. The coffe service is located at LabTwo which is subsumed by BuildC. These identifications are necessary because, in our model, the distances are defined between instances of *FixedStructure* (buildings). The localization agent's returns the distance which is added to the belief base of the user_interface agent. Then the user_interface agent shows the distance to the user.

5 Conclusions

In this paper we have shown a practical prototype that integrates BDI agents with the Semantic Web framework Jena. This integration was proposed to deal with the dynamic nature of mobile computing applications. Agents in the environment in which the user is located communicate with the user personal agent to inform about locally situated services. Contact with the user can be triggered by matching the user profile and context description. Ontologies are well suited for providing such profile and context descriptions. They represent the required knowledge in a structured and organized way, allowing inference for integrating user's goals with the context features. Also, this knowledge can be both queried and modified by agents.

In [5] AgentSpeak with underlying ontological reasoning was first proposed and formalized. That extension was shown to have the following effects: (i) more expressive queries to the agent belief base; (ii) refined belief update, new beliefs can only be added if the resulting belief base is consistent with the concept description; (iii) more flexible plan search based on the subsumption relation between concepts; and (iv) knowledge sharing by using web ontology languages such as OWL. In that paper, it was shown how extending an agent programming language with the descriptive and reasoning power of description logics can have a significant impact on the way agent-oriented programming works in general and in particular for the development of Semantic Web applications using the agent-based paradigm. However, the practical development of that previous proposal would require changes in the current AgentSpeak framework (Jason).

The Jason/Jena integration proposed and illustrated here is, of course, a much simpler approach from what was proposed in that work. However, one advantage of our proposal is that it allows the use of OWL ontologies without any modification or redefinition in the currently available frameworks. The prototype has shown the main components of such an architecture, which can be exploited for more elaborated applications in future work. For example, we haven't fully explored inference and

[3] http://www.w3.org/Submission/RDQL/

reasoning that is provided by the Jena framework. For now we have only used RDQL to query an OWL knowledge basis. In fact, with this first prototype we have only accounted for the improvements referred to in (i) and (iv) above. We believe that points (ii) and (iii) could as well be pursued through a deeper integration of the technologies that we have adopted here.

Our prototype implements a fairly simple application, which serves mainly to the purpose of illustrate the potentiality that these technologies bring about when they are put together, being a first practical approach integrating OWL, Jason and Jena. For now, when executing the Argonaut, the presence of a user is simulated. Ideally, that is, in a real implementation, this perception would happen through sensors located at different locations. Also, in a more sophisticated implementation, the agents could be distributed over a network. The View module could be customizable based on information retrieved from the ontology, in a contextualized way, considering the user profile and device characteristics. For this, an ontology extension would be required. Other similar interesting applications could be explored for the presented prototype, such as, for instance, cars communicating with available services in the road.

References

1. Bordini, R., Hubner, J. and Wooldridge, M.: Programming AgentSpeak Agents with Jason. John Wiley & Sons. 288p (2007).
2. Chen, H., Chen, H., Perich, F., Finin, T., Joshi, A: SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications, In Proceedings of the First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (Mobiquitous 2004), Boston, MA, August (2004).
3. Chen, H., Chen, H., Perich, F., Chakraborty, D., Finin, T., Joshi, A.: Intelligent agents meet the semantic web in smart spaces. *IEEE Internet Computing*, 19(5):69–79, November/December 2004.
4. Horridge, M., H. Knublauch, A. Rector, R. Stevens, C. Wroe.: A Practical Guide To Building OWL Ontologies Using the Protégé-OWL Plugin and CO-ODE Tools, Technical Report, Ed. 1.0, The University Of Manchester (2004).
5. Moreira, A., Vieira, R., Bordini, R., Hubner, J.: Agent-Oriented Programming with Underlying Ontological Reasoning. In: Declarative Agent Languages and Technologies III: Third International Workshop, Utrecht, The Netherlands, Selected and Revised Papers. Vol. 3904, pp. 155--170. Springer, Berlin (2006)
6. Padgham, L. and Winikoff, M.: Prometheus: A Methodology for Developing Intelligent Agents. LNCS, vol. 2585, pp. 174--185. Springer (2003)
7. Staab, S. and Studer, R. (eds.): Handbook on Ontologies. International Handbooks on Information Systems. Springer-Verlag, Berlin–Heidelberg (2004)
8. Vieira, R., Moreira, A., Bordini, R. and Wooldridge, M.: On the Formal Semantics of Speech-Act Based Communication in an Agent-Oriented Programming Language. Journal of Artificial Intelligence Research, Vol 29, p. 221-267 (2007)