# Using Agents and Ontologies for Application Development on the Semantic Web

**Marcelo Blois**[1]**, Maurício Escobar**[1] **& Ricardo Choren**[2]

[1] FACIN, PUCRS
Av. Ipiranga, 6681, Porto Alegre/RS, 90619-900 - Brazil
{blois, escobar}@inf.pucrs.br

[2] Computer Engineering Department, IME
Pça Gen Tibúrcio, 80, Rio de Janeiro/RJ, 22290-270 - Brazil
choren@de9.ime.eb.br

## Abstract

*The Semantic Web provides access to heterogeneous, distributed information, enabling software products to mediate between user needs and the information sources available. Agents are one of the most promising technologies for the development of Semantic Web software products. However, agent-based technologies will not become widespread until there are adequate infrastructures for the development of semantic multi-agent systems (MAS). Some challenges, such as turning software agents into practical abstractions for dealing with ontologies, taking advantage of the distributed nature of the Web to create distributed agents and making a seamless integration with existing Web tools, e.g. the browser, still need to be addressed. This paper describes the main features of the SemantiCore framework, an agent infrastructure to develop semantic MAS. A look at a benchmark Semantic Web application illustrates the SemantiCore potential as an infrastructure for the deployment of semantic agent applications.*

***Keywords:*** multi-agent systems, semantic Web, agent infrastructure, ontology.

## 1. INTRODUCTION

The ever-increasing importance of the Web in everyday life is driving the need for software capable of coping with open and dynamic environments [1]. Indeed, networking systems and Web technologies is fostering the use of the Internet as a basis for software development. More than other technologies, software agents seem to have the necessary characteristics to support the development of open and flexible software systems [11, 19, 22, 23] such as Web-based systems.

Multi-agent systems (MAS) are appropriate for domains that are naturally distributed. The use of agent concepts for distributed systems engineering provides several advantages for reducing complexity, including [12, 20]: autonomy, situatedness and high-level interactions. Interaction between agents enables the construction of a community of software programs. Without some shared or common knowledge, the agents of MAS have little hope of effective communication [15, 16]. Thus the design of agent systems require [24]: suitable design abstractions to support shared knowledge exhibited by agent societies, i.e. knowledge that cannot directly be ascribed to individual agents [4, 25]; and, suitable infrastructures that semantically shape the agent environment to enable and promote the exploitation of this knowledge.

In this sense, the Semantic Web is a platform in which information is given well-defined meaning, better enabling

knowledge representation, and allowing agents to create the social intelligence. Particularly in the most common way to solve the problem of enabling software agents to interoperate over the semantic Web is to give each of them the same specified conceptualization, or ontology, of the domains they are expected to work in. It is important to mention that this ontology can be public information or an agreed set of definitions and meanings of basic communicable concepts, i.e. information that was not develop specifically for a given application.

Thus it would be desirable for agent infrastructures to support the development knowledge-aware systems so that agents can collect Web content (shared knowledge) from diverse sources, process the information and exchange the results with other agents. Other challenging problem in the deployment open and flexible systems using the Web infrastructure is the development of agent platforms that provide seamless distribution and integration with existing Web tools, such as the browser. An agent platform for the Semantic Web should take advantage of the Web infrastructure itself so that agents interact with other agents in an open environment, not limited to an agent container. Moreover, the user entry-point for the Web usually is a browser application. Thus agents should be integrated with browsers to support user access to online information and functionalities.

These issues are not directly addressed by the existing multi-agent development platforms (e.g. [9, 21]). These platforms were created for the development of closed agent systems, i.e. systems in which agents are encompassed within the boundaries of a specific container (limited environment), not using the (semantic) Web infrastructure as a basis. If agents are to keep their promise as a technology for semantic Web application development, it makes no sense to have two disjoint concepts: the agent container and the Web.

The aim of this paper is to present SemantiCore, an agent infrastructure that integrates the richer semantics of the Semantic Web to the implementation of agent systems. SemantiCore is a framework that provides an abstraction layer for agent-oriented application development for the Semantic Web. It can be integrated with the current web infrastructure extending its computational capabilities to allow agents to process semantic content while the user is navigating on annotated web pages.

This paper is structured as follows. In section 2, we give an overview of the role of software agents in the Semantic Web. From this overview, we show that agents are suitable to develop Semantic Web applications. Section 3 introduces the SemantiCore framework. It describes the agents the framework provides, the semantic agent lifecycle and its components. Section 4 illustrates how the framework is successfully exploited in a benchmark Semantic Web application. Section 5 shows some related work and, finally, we draw conclusions and look at challenges for future research on environments in section 6.

## 2. AGENTS AND THE SEMANTIC WEB

Semantic Web technologies seam to be way to provide the semantic integration between data and processes across systems that can be owned by different enterprises [2]. This technology is still in progress [3, 8], for instance the definition of languages for expressing the semantics of the Web is not mature yet [14]. However different the powerful synergism between agents and Semantic Web could be very promising [13, 18] and some efforts have been made in order to define ontology models and develop tools suitable for agents aiming at being truly semantic-aware agents.

Therefore a key component in the semantics-rich approach is the ontology. An ontology is the specification of a conceptualization, that is it is the formal, agreed vocabulary whose terms are used in the construction of a semantic system. An ontology is a conceptualization of an application domain in a human-understandable and machine-readable form, and typically comprises the classes of entities, relations between entities and the axioms which apply to the entities which exist in that domain.

The Semantic Web will not be primarily comprised of nice organized ontologies that have been carefully constructed by experts; instead, it will be a complex web of semantics ruled by the same sort of havoc that currently rules the rest of the web. Rather than a few large, complex, consistent ontologies, shared by great numbers of users, the Web will be composed of a great number of small ontological components [7]. Information will be exchanged between applications, allowing computer programs to collect and process web content, and to exchange information with each other.

An agent is a software system that is (i) situated in some environment, (ii) capable of autonomous actions in order to meet its objectives and (iii) capable of communicating with other agents [20]. Therefore agents are situated systems and the environment should provide agents the essential information to work. Thus the highly semantic infrastructure of the Semantic Web will make agent-based computing much more practical [7].

However, one challenge for Agent-oriented Software Engineering (AOSE) is to provide an affordable way to introduce the design and implementation of intelligent (knowledge-based) behavior into mainstream Software Engineering [24]. The issue is not simply provide a method for integrating ontologies and agents but providing an implementation framework that is not simply creating dedicated (isolated) containers for agents to execute but rather use the Semantic Web and all its features to deploy open and flexible software systems composed of knowledge-aware agents.

## 3. THE SEMANTICORE FRAMEWORK

MAS development platforms usually focus on the distribution issues such as concurrency control, message passing, environment management and internal agent architecture implementation. SemantiCore was developed to fulfill the gap the existing MAS platforms have for creating open MAS on the Web capable of processing semantic content annotated in Web pages. SemantiCore was developed to be integrated to web servers and web browsers in order to create Semantic Domains where software agents live thus extending the Web without interfering with its current structure. These Domains turn the current request-response paradigm used on the Web into a hybrid request-response / peer-to-peer model.

Agents living on the client's machine in a Semantic Domain associated to a web browser can contact other agents in other Semantic Domains. Thus, a Semantic Domain can be defined as an extension of the current web domain that provides an environment for agents capable to process ontologies to run. This hybrid computational model allows agents to coordinate their actions to other agents "living" on the web to achieve a common goal. Figure 1 illustrates how SemantiCore modifies the current web architecture providing a software layer that processes semantic content using intelligent agents.

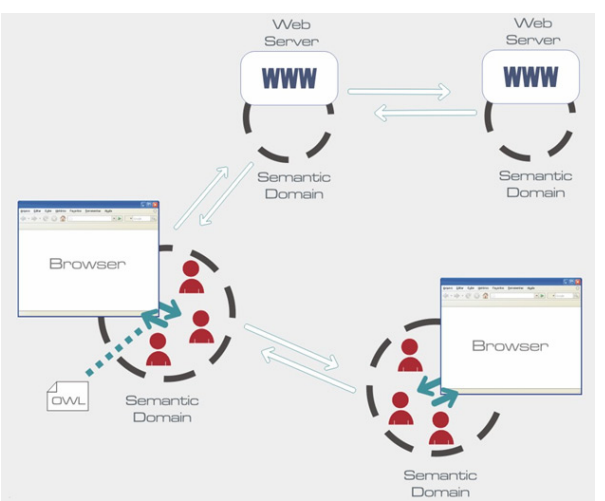


Figure 1: SemantiCore integration with the current web infrastructure.

Since SemantiCore is based on the FIPA Reference Model [6] it provides internal domain messages routing, external domain messages routing, services directory facilities, agent registration services and agent execution contexts. These characteristics are distributed among different domain agents such as the Environment Manager, the Domain Controller and the Service Directory.

The Environment manager is an agent that captures information on the web browser or the web server, depending if the Semantic Domain is integrated to a browser or a web server, and passes this information to the Semantic Domain. It also passes back the information resulted from the agents processing to the web. This is a central feature since the ontologies that are captured by the browser must be passed to the agents in the Semantic Domain and the results of the agent's action may interfere on the browser navigation and content presentation. When the user navigate in a Semanticore enabled domain the server must send response variable in the HTTP protocol to signal the presence and the identifier of the Semantic Domain. The Semantic Domain attached to the browser is then able to connect the Semantic Domain attached to the server allowing agent to communicate directly through the environment.

The Domain Controller has, among other capabilities, the potential to connect different Semantic Domains depending on the interests of the agents running on these domains. It is responsible to detect and connect different Semantic Domains allowing the message passing and routing to the appropriate receiver. It also maintains regulations over the agents' actions related to environment resources usage and code migration (for mobile agents). Social regulations and agent specific regulations must be implemented directly by the developer and do not use the supervisory feature provided by the Domain Controller.

Other Semantic Domains features include the support of native code migration, which allows the creation of mobile agents and the definition of different channels for message exchange. Migration can be done simply by a method invocation which must indicate the location to move. Control messages among management agents and components of an agent are sent in a separate control channel while messages exchanged by application agents are sent using a data channel. This provides control message isolation and less interference of the domain's control-related traffic on application agent communication.

### 3.1. THE SEMANTICORE AGENT LIFECYCLE

SemantiCore agents extend the Semantic Agent class (figure 2). The agent starts its execution by calling the setup method. During setup, the developer may create sensors, facts, rules, effectors, actions, action plans, and goals for the agent. All these structures are created using SemantiCore classes and form the SemantiCore reference model for the Semantic Agent. Figure 3 shows an example code extracted from an agent responsible to answer other agent's requests on lists of clinics suitable for a certain type of physiotherapy treatment.

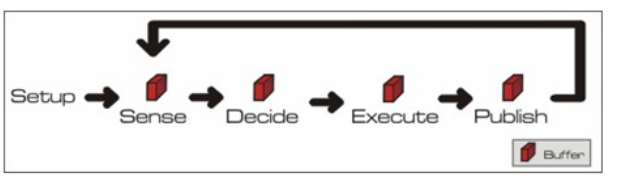


Figure 2: The SemantiCore agent lifecycle.

```
1.  protected void setup ()
2.  {
3.    addSensor ( new TestSensor ( ) );

4.    fullModel = ModelFactory.createOntologyModel ( );

5.    OntModel clinicSchema = OWLUtil.ontModelFromFile ( "./Schemas/clinic.owl" );

6.    fullModel.add ( clinicSchema );

7.    setDecisionEngine ( new InferenceJenaEngine ( fullModel ) );

8.    try
9.    {
10.     SimpleFact sf1 = new SimpleFact ( "?clinic", "rdf:type", "http://semanticore.pucrs.br#Clinic" );
11.     SimpleFact sf2 = new SimpleFact ( "?clinic", "http://semanticore.pucrs.br#distance", "?dist" );
12.     SimpleFact sf3 = new SimpleFact ( "?clinic", "http://semanticore.pucrs.br#qualification", "'excellent'" );

13.     FunctionBasedFact fbf = new FunctionBasedFact ( "lessThan", new String [ ]{ "?dist", "20" } );
14.     ComposedFact cFact1 = new ComposedFact ( sf1, sf2 );
15.     ComposedFact cFact2 = new ComposedFact ( cFact1, sf3 );
16.     ComposedFact cFact3 = new ComposedFact ( cFact2, fbf );

17.     SimpleFact cSf1 = new SimpleFact ( "?clinic", "http://semanticore.pucrs.br#status", "'true'" );
18.     ReceivedClinicsAction aClinic = new ReceivedClinicsAction ( "Clinics", new String [ ]{ "?clinic", "?dist" } );
19.     ComposedFact clinicConsequence = new ComposedFact ( aClinic, cSf1 );

20.     Rule clinicRule = new Rule ( "rule1", cFact3, clinicConsequence );

21.     addRule ( clinicRule );

22.   }
23.   catch ( Exception e )
24.   {
25.     e.printStackTrace ( );
26.   }

27. }
```

Figure 3: A setup method example

The Semantic Agent class has some in-built methods to add all the elements an agent may have in SemantiCore. The addSensor method enables the agent to add a previous defined TestSensor in its definitions (line 03). Sensors are created extending the Sensor class. Agents may have different types of sensors working simultaneously depending on the kind of messages they want to capture, e.g. OWL or SOAP messages. The setDecisionEngine (line 07) method is used to define the inference engine to be used for manipulating ontologies. This method allows the creation of agents with different decision making methods, e.g. inference engines, neural networks and decision trees.

During setup it is also possible to define facts, rules and actions. SimpleFact, FunctionBasedFact and CompositeFact classes are used to create different patterns of facts that may be associated to the decision making method or that may be part of a rule this method must take into account. An action extends a FunctionBasedFact so it may be automatically started from the decision making method (in this case the inference engine). The action has some arguments which are used to pass the data sensed in the environment or discovered during the decision making. The developer has complete access to the agent and its execution context (such as variables) in each action.

Setup is only executed once when the agent is created in the environment. SemantiCore's agents are instantiated and registered by the Domain Controller agent. Once an agent has started, it basically performs 4 operations in loop during its execution: senses the environment, decides in accordance to the information sensed, executes the actions depending on the decisions made and publishes information back in the environment. This lifecycle is automatically managed by SemantiCore.

### 3.2. THE SEMANTICORE AGENT COMPONENTS

The four basic lifecycle operations are encapsulated into components: sensorial, decision, executor and effector. These specialized components allow better maintenance, extensibility and organization of code. Figure 4 shows the agent component architecture. The Sensorial component manages all the different sensors an agent has, selecting one of them dynamically according to the kind of message received from the environment. This component generates a list of objects that are transmitted to other components depending on the communication links mapped. The basic SemantiCore distribution maps the sensorial output to the decision input.

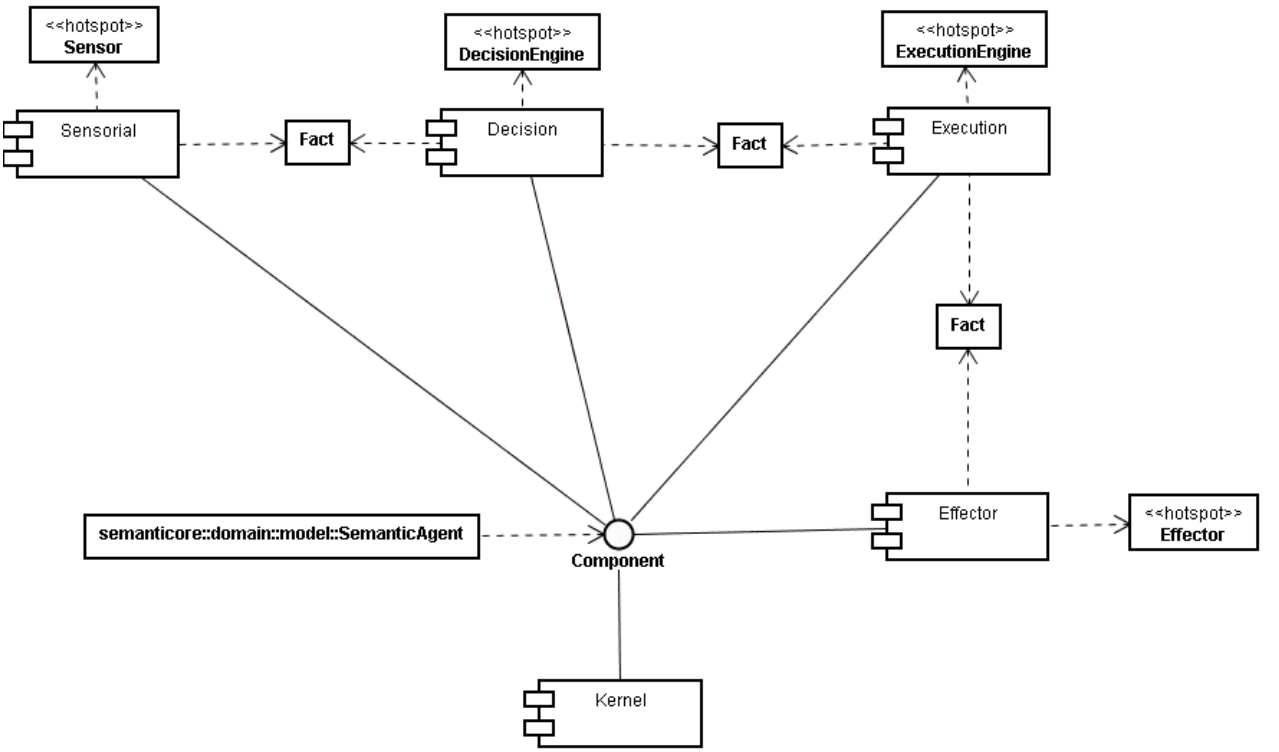


Figure 4: The agent component architecture.

The Decision component handles the rules and facts that form the mental model of an agent. It also manages the decision making mechanism used to decide over the data received by sensorial output. The facts and rules have specific hotspots for their representation in different formats. The basic SemantiCore distribution already has hotspots implementation to codify the rules and facts in the format accepted by the Jena's OWL Reasoner. The output of the decision making is a list of actions to be performed.

The Execution component manages the execution of agent actions allowing developers to access the data internally stored by the agent and all the agent's primitives. Semantic agents have all information sensed stored in their data contexts. This information is formed by individuals in the ontologies processed and it is made available to all running actions by the execution component.

Actions may need to send messages to other agents in the Semantic Domain as a part of its processing. The encapsulation of these messages in different structures and their transmission in the environment is the responsibility of the Effector component. The effector component selects dynamically the effector that must be used to codify the message that will be transmitted. This is done by a dynamic instantiation depending on the type of effector needed to send a message. For example, if the message must be sent using an OWL effector, then the OWLEffector class will be dynamically instantiated. When an agent is created, the developer may indicate the types of effectors the agent will work with. This feature allows an agent to talk simultaneously with different peers such as web services (SOAP messages), other agents created in a FIPA-compliant platform (ACL messages) or other SemantiCore agents (OWL messages).

### 3.3 SEMANTICORE AND THE SEMANTIC WEB

Component organization, different hotspots instantiation, different formats and native OWL processing contribute to differentiate SemantiCore from other MAS platforms. SemantiCore enables the agent component distribution over machines. This unique feature allows SemantiCore to adapt to particular Semantic Web performance requirements. For example, if the Decision component demands higher processing capability than the Sensorial component, it may be distributed in a high performance hardware.

Figure 5 shows the result of a performance test executed on an application developed (see section 4) using SemantiCore. The graphics show the performance upgrade distribution. Notice that the average performance upgrade is 16,66% for distributed decision execution. The performance upgrade continues to be relevant as the number of agents in the domain grows although it decreases in absolute values. This is partially explained by the overhead produced by multiple threads management on a single machine versus the communication overhead among them.

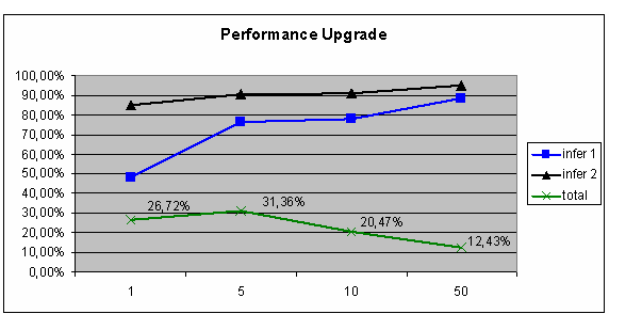


Figure 5: Performance upgrade using SemantiCore distribution.

The test (repeated 5 times) was run using four Pentium IV, 3.4 Ghz, duo core, 1Gb RAM, running Windows XP operating system, and connected in a 100Mbps wired network. Each machine had a part of the same SemantiCore domain where 4 agents were executed. One of these agents had the decision component centralized in the first testing set and distributed in the second one.

Another relevant characteristic is SemantiCore ontology-based agent representation as shown in figure 6. This representation maps all the elements a Semantic Agent has, allowing it to reason about itself and about other agents in the domain. This representation also allows browsers to capture agent definitions on Semantic enabled web sites and execute the agent based on the representation. It is important to consider that the ontology does not have only concept definition, but also instances definition enabling the agent instantiation. The content of the agent action is coded using the Java language embedded in the instance ontology definition. Other elements use the ontology properties to define their instances.

## 4. A SAMPLE MAS USING SEMANTICORE

This section presents a case study adapted from the classical Semantic Web example published in [2]. The adaptation aimed to better specify the problem in order to build an agent-based solution for this problem.

### 4.1. PROBLEM DESCRIPTION

The example shows two brothers, Pete and Lucy, trying to schedule physiotherapy sessions for their mother, who will be called Marie for explanations purposes. Pete and Lucy have personal digital assistants (Semantic Agents, named AgPete and AgLucy respectively) to execute certain specialized tasks. One of such tasks is the sessions' scheduling. The discussion about how they interface with their users is out of the scope of this work.

The first interaction occurs when Lucy decides to give the scheduling task to her agent. AgLucy starts the execution of its action plan specially developed for this purpose which comprehends the following tasks: (i) recover Marie's prescribed treatment with the doctor's personal digital assistant; (ii) look for the clinics that offer this kind of physiotherapy sessions; (iii) rank clinics that are in-plan for Mom's insurance within a 20-mile radius of her home and with a rating of excellent or very good on trusted rating services; (iv) send the searching results to Pete's agent; (v) negotiate available time frames for driving Marie to the clinic with AgPete; and (vi) fix the appointments on her personal agenda. Due to space restrictions this section will only show AgLucy's code comprising steps (i) to (iv).

### 4.2. THE APPLICATION AGENTS

The multi-agent system developed to solve the previous problem is composed by the Health care service catalog agent (HealthCareAgent), the doctor agent (AgDrLee), the trust rating service agent (ClinicsChecker), and Lucy's and Pete's agents. These agents execute on different machines and can and may be on a single domain or multiple domains since SemantiCore abstracts the distribution issues. Figure 7 illustrates the solution scenario.

The HealthCareAgent implements a health care service provider catalog, executing searches in it catalog based on other agents' requests. Requests are done using keywords represented in an ontology. The

agent tries to find the most appropriate service description according to the query ontology. The similarity between ontologies is an open issue and this was implemented in this example using an adaptation of OntoMetric [22]. This agent only has one kind of Sensor which is an instantiation of the regular OWLSensor provided by SemantiCore. It has a very simple decision capability that maps each query fact to an action execution. This action is implemented by the FetchClinicsAction class. It encapsulates the heuristic just described and publishes in the environment the query results using the regular OWLEffector provided by SemantiCore.

The AgDrLee agent main goal in the example is to inform patient's treatments based on his medical records. To achieve this goal, the agent must receive a retrieve treatment message with a name of one of its patients. Then it decides, based on the patient's characteristics, in which record to look for. This is achieved using simple decision rules in the decision component. Once a record is found, the agent starts the appropriate action plan RetrieveTreatmentInformation. The information retrieved is associated to a predefined OWL schema as instances of its classes and published using the regular OWLEffector.

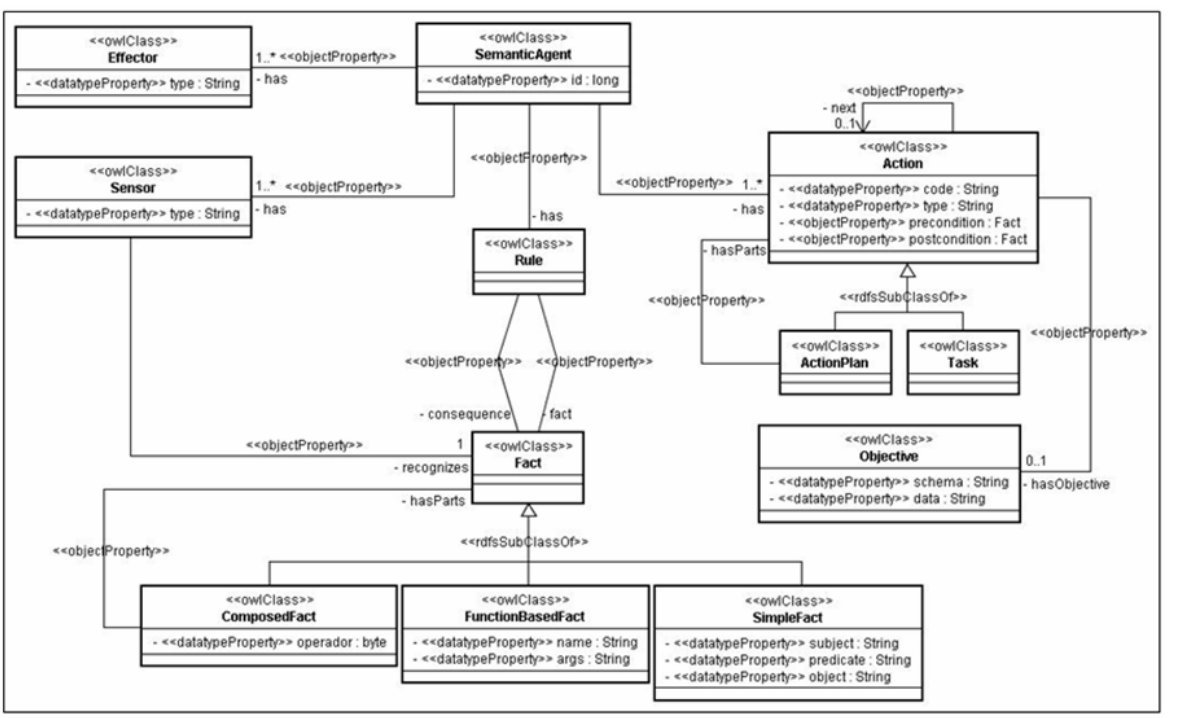


Figure 6: Ontology-based agent representation.

The ClinicsChecker agent certificates different types of medical service providers based on predefined quality of service evaluations, allowing other agents to query for certificates for heath care service providers. This agent was implemented with a single Sensor to capture requests based on the clinic's medical registration. The clinic information is passed to the decision engine and then to the execution component to be processed. This initiates the action plan RetrieveClinicsQualification. This plan queries a database for the clinic certificate and possible incidents it may be involved in.

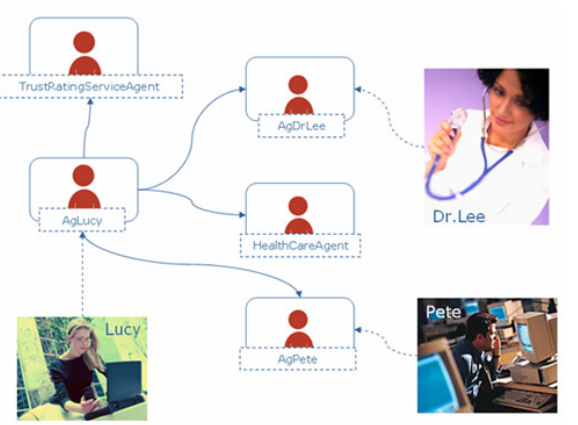


Figure 7: The sample application solution scenario.

AgLucy is Lucy's personal agent and it encapsulates all the necessary elements to coordinate different requests and decide if a clinic is suitable for scheduling physiotherapy sessions or not, based on Lucy's and Pete's constraints. Since this agent is central in our example, its code will be presented in detail to illustrate how agents were implemented.

The agent definition begins in its setup method where 3 action plans are declared: RetrieveTreatment, RetrieveClinicsList and RetrieveClinicsQualification. The first allows the communication with the AgDrLee to retrieve Marie's prescribed treatment. The second contacts the HealthCareAgent to recover the list of clinics which provide physiotherapy sessions. The later contacts the ClinicsChecker in order to rate the clinics in terms of quality of service. These action plans are associated with the agents' goals and must be executed when their pre-conditions are achieved and/or the agent inference indicates so. Some rules and facts are created to map these pre-conditions and inference chains. For instance, AgLucy has some facts and rules that relate the distance and the qualification of a Clinic to its selection from the list as a consequence of then inference processing.

In the setup the agent also has the hotspot instantiation indicating which decision engine will be used. In this example, AgLucy indicates through the setDecisionEngine method the use of the InferenceEngine class as the hotspot implementation. This hotspot implementation is distributed with SemantiCore and integrates Jena's inference engines into the Semantic Agents. The rules and facts defined in setup are translated automatically to the format used by the inference engine.

Other interesting SemantiCore's feature that is largely used in this example is the automatic variable definition and attribution based on the data sensed capability. When someone declares a fact which refers to ontology concepts such as http://semanticore.pucrs.br#distance SemantiCore creates a variable with the same name in the agent context. This variable will have its value automatically set when an ontology arrives through the agent's sensor with individuals for the concept defined. The developer can access these values directly in the action definition of an action plan.

AgPete follows the same structure already presented for AgLucy. The main difference is the strategy used in its action plans to achieve the goal. AgPete does not trust in the HealthCareAgent search capability and thus it decides to fetch the clinics querying AgDrLee for indications. With a new candidate list, it checks with the HealthCareAgent if the service provider is registered in the health care plan of Pete's mother. Finally it executes the same trust rating service checking to find the clinics ratings and decides which is the most appropriate.

### 4.3. COMMENTS ABOUT THE IMPLEMENTATION

The system was implemented using two different interfaces. The first implementation used an interface that allowed Lucy to control her agent through a handheld. The other implementation used a simplified browser written in Java that was integrated to SemantiCore. This browser captured all the OWL annotation indicated in a web page and encapsulated it on a semantic message. It transmitted the semantic message through the Environment Manager to the agents in the local Semantic Domain. If an agent had a sensor programmed to capture the kind of information represented in the ontology, it started the information processing. The processing results were delivered back to browser by the Environment Manager. In the last scenario the AgLucy interaction with the user was completely done using the regular web interface. The other agents executed in Semantic Domains integrated to web servers while AgLucy and AgPete executed on the client's machines.

## 5. RELATED WORK

Some works aim to support the creation of semantic Web applications, such as Jena [10] and OntoBuilder [17]. The Jena Semantic Web Toolkit is a Java Application Programming Interface (API) and software toolkit for manipulating RDF, RDFS, OWL and SPARQL and includes a rule-based inference engine. Using Jena it is possible to manipulate, query and store OWL files and to create new inference engines. OntoBuilder supports the extraction of ontologies from Web search interfaces, ranging from simple search engine forms to multiple-pages, complex reservation systems.

There are no agent platforms specifically designed for supporting the deployment of agent systems in the Semantic Web. Jade [9] is an agent framework implemented in Java language that supports the implementation of multi-agent systems through a middleware that complies with the FIPA specifications. Jade provides some Java classes for ontology manipulation, which allows the development of agents that can use ontologies as objects. Nevertheless, Jade agents are not specifically designed to take advantage of the Semantic Web. For instance, agents in a MAS execute in a container that are separated from the containers of other MAS applications.

## 6. CONCLUSIONS

Semantic content automatic interpretation is still an open issue for Semantic Web researchers. Besides issues related to ontology mapping and similarity

there are other important issues such as how we can leverage the full potential of the Semantic Web to our daily applications that must be addressed before the fully adoption of the Semantic. The Semantic Web uses different protocol layers that turn the application development a hard exercise. A high level abstraction must be used to ease application development for the Semantic Web.

This paper presented the SemantiCore framework which aims to provide an abstraction layer developers can use to create their semantic application without having to deal with all the implementation details involved in this task. SemantiCore uses the software agent as the basic abstraction due to its application in the solution of distributed complex problems. SemantiCore can be integrated in the current web infrastructure to allow agents to execute on semantic domains associated with the regular web browsers and servers.

SemantiCore agents have the ability to reason about themselves and the others due to the ontological representation of each Semantic Agent. Agents are created using the basic Semantic Agent elements and their lifecycle are managed automatically by the Semantic Domain.

SemantiCore was used in practice to develop 3 different complete applications (a Conference Management System as defined in [5], an automatic advertisement system which offered special prices on meals based on the customers' (represented by agents) food preferences and their geographic location in relation to a restaurant (also represent by an agent); and a MAS platform with different hotspots instantiation to develop an Intelligent Computer Integrated Manufacturing system for a 5-robot-station production cell.) and the case study presented in this paper which allows the critical analysis of its features.

This paper also showed some preliminary SemantiCore performance testing results. Although it is appropriate for Semantic Web application development, SemantiCore has some limitations and improvements opportunities. For instance, a visual agent development tool must be provided in order to facilitate the task since the agent is formed by a relatively big set of elements. This visual composer may also have intrinsic ontology development support so the facts and rules can be directly implemented and tested using ontologies. Also, it is necessary to extend SemantiCore to integrate it with well-known web browsers and servers such as Mozilla Firefox and Apache Web Server. This can be done by creating a SemantiCore plug-in for them. A SemantiCore-enabled Firefox browser is currently under development. This extension will turn the Semantic Web navigation completely transparent to the user turning the Semantic Web ideas into reality.

## REFERENCES

**[1]** Bergenti, F.; Poggi, A. Agent-oriented software construction with UML. *The Handbook of Software Engineering and Knowledge Engineering (vol. 2)*, Emerging Technologies, 2002, pp. 757-769.

**[2]** Berners-Lee, T.; Hendler, J.; Lassila, O. The Semantic Web, *Scientific American 1(5)*, 2001, pp. 34-43.

**[3]** de Bruijn, J.; Polleres, A.; Lara, R.; Fensel, D. OWL DL vs. OWL Flight: Conceptual Modeling and Reasoning for the Semantic Web. *Proceedings of the 14th International World Wide Web Conference*, 2005, pp. 623-632.

**[4]** Ciancarini, P.; Omicini, A.; Zambonelli, F. Multiagent systems engineering: The coordination viewpoint. *Intelligents Agents VI: Agent Theories, Architectures, and Languages*, LNAI 1767, Springer-Verlag , 2000, pp. 250–259.

**[5]** DeLoach, S. A. Modeling organizational rules in the multi-agent systems engineering methodology. *Proceedings of the 15th Congress of the Canadian Society for Computational Studies of Intelligence*, LNCS 2338, 2002, pp. 1-15.

**[6]** FIPA ACL Message Structure Specification. *http://www.fipa.org/specs/fipa00061*, 2001.

**[7]** Hendler, J.A. Agents and the Semantic Web, *IEEE Intelligent Systems 16(2)*, 2001, pp. 30-37.

**[8]** Horrocks, I.; Patel-Schneider, P.F. A proposal for an OWL rules language. *Proceedings of the 13th International World Wide Web Conference*, 2004, pp. 723-731.

**[9]** Java Agent DEvelopment Framework. *http://jade.tilab.com/*, 2006.

**[10]** Semantic Web Framework for Java. *http://jena.sourceforge.net/*, 2006

**[11]** Jennings, N.R.; Wooldridge, M. Agent oriented software engineering. *The Handbook of Agent Technology*, MIT Press, Massachussetts, 2000, pp. 1-24.

**[12]** Jennings, N.R. An agent-based approach for building complex software systems. *Communications of the ACM 44(4)*, 2001, pp. 35–41.

**[13]** Labrou, Y. Agents and ontologies for e-business. *Knowledge Engineering Review*

*17(1)*, 2002, pp. 81-85.

**[14]** Negri, A.; Poggi, A. ; Tomaiuolo, M.; Turci, P. Agents for e-Business Applications. *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems*, 2006, pp. 907-914.

**[15]** Lister, K.; Sterling, L. Agents in a Multi-Cultural World: Towards Ontological Reconciliation. *Proceedings of the 14th Australian Joint Conference on Artificial Intelligence (LNCS 2256)*, 2001, pp. 321-332.

**[16]** Lister, K.; Sterling, L. Reconciling Ontological Differences for Intelligent Agents. *Proceedings 5th International Joint Conference on Autonomous Agents and Multiagent Systems*, 2006, pp. 943-945.

**[17]** OntoBuilder, *http://iew3.technion.ac.il/OntoBuilder*, 2006

**[18]** Parunak, H.V.D. Go to the ant: Engineering principles from natural agent systems. *Annals of Operations Research 75*, 1997, pp. 69–101.

**[19]** Patil, R.S.; Fikes, R.E.; Patel-Scheneider, P.F.; McKay, D.; Finin, T.; Gruber, T.; Neches, R. The DARPA knowledge sharing effort: progress report. *Proceedings of 3rd Conference on Principles of Knowledge Representation and Reasoning*, 1992, pp. 103-114.

**[20]** Silva, N.; Rocha, J.; Cardoso J. E-Business interoperability through ontology semantic mapping. *Proceedings of Processes and Foundations for Virtual Organizations*, 2003, pp. 315-322.

**[21]** Sycara K.P.; Paolucci, M.; van Velsen, M.; Giampapa J.A. The RETSINA MAS Infrastructure. *Journal of Autonomous Agents and Multi-Agent Systems 7(1-2)*, 2003, pp. 29-48.

**[22]** Tello, A.L.; Gómez-Pérez, A. ONTOMETRIC: A Method to Choose the Appropriate Ontology. *Journal on Database Management 15(2)*, 2004, pp. 1-18.

**[23]** Wooldridge, M. Agent-based software engineering. *IEE Proceedings on Software Engineering 144(1)*, 1997, pp. 26-37.

**[24]** Zambonelli, F.; Omicini, A. *Challenges and Research Directions in Agent-Oriented Software Engineering, Autonomous Agents and Multi-Agent Sytems 9*, 2004, pp. 253–283.

**[25]** Zambonelli, F.; Jennings, N.; Wooldridge, M. Developing multiagent systems: The Gaia methodology. *ACM Transactions on Software Engineering Methodology 12(3)*, 2003, pp. 417–470.