Semantic Web Communication Framework Towards Agent-based Software Engineering

Yong-Feng Lin and Jason Jen-Yen Chen
Dept. of Computer Science and Information Engineering,
National Central University,
Jhong-Li, Taiwan

ABSTRACT

This work proposes an innovative framework towards realizing agent-based software engineering. As a new abstraction, agents reveal significant potential to intimately serve human users using web services. This framework coordinates web services over the Semantic Web. It adapts agent communication to the Semantic Web, then confines inter-agent communication within the foundation of intelligent and physical agents (FIPA) interaction protocol. Through this, automated web service discovery and composition is expected to be realized in the near future.

Key words
Agent, Semantic Web, web service, foundation of intelligent and physical agent (FIPA)

1: INTRODUCTION

With the paradigms shift depicts in Fig. 1, software engineering keeps moving forward. The *object* paradigm models software components as real-world entities. It encapsulates functions and data to obtain classes that instantiates objects. It argues that running software is comprised of a collection of objects, as opposed to the traditional *function* paradigm in which a program is seen as a collection of functions.

The World Wide Web (WWW) attracts more and more people to use computer then ever before. Most of them are casual users. The object paradigm, however, lacks the mechanism to model casual users' experiences, attitudes and thoughts in a distributed manner, although it is good at modeling software as just said.

The recent Semantic Web explores a new vision of the Web. One of its efforts is the web ontology language for web services (OWL-S) [1]. OWL-S provides a computer-interpretable service description for agents [2] [3] to integrate a variety of web services [4] over the Semantic Web. With the distributed nature of those services, the idea of agent-based software engineering (ABSE) has emerged. The *agent* paradigm argues that a program is to manipulate agents' behaviors. Agent contains *modal* operators to infer user's mental attitudes that drive program execution.

This work presents a communication framework that provides a modal-driven interaction for the agent paradigm that, in a sense, corresponds to the message passing mechanism for the object paradigm, or the function call mechanism for the function paradigm.

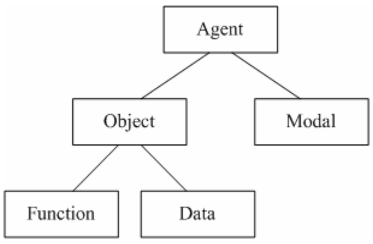

Fig.1: Software engineering paradigms shift: function, object and agent

This framework provides a set of markup language constructs describing the foundation of intelligent and physical agents (FIPA) interaction protocol in computer-interpretable form. Moreover, it provides a mental model for agents to drive the interaction protocol in formal semantics. Inter-agent communication distributed over the Semantic Web has to conform to the same interaction protocols. Thus, agents can coordinate web services over the Semantic Web (semantic web service [5]) in a standardized manner.

2: RELATED WORK

Three agent frameworks: Jfipa [6], Jade [7] and Cougaar [8], are surveyed. Jade and Cougaar are built in their respective development environments. This section compares these frameworks from the perspectives of: 1) agent adaptivity over the Semantic Web; 2) agent interoperability over the agent platform and 3) integrativity of agent and the Semantic Web.

1. Agent adaptivity over the Semantic Web

In the context of the Semantic Web, ontology provides an environment to share understanding of a domain [9]. Adapting an agent's native ontology to the Semantic Web's ontology enables the Semantic Web to be observed by agents. Jfipa provides a FIPA XML-based message routing mechanism, but does not support ontology-based communication. Jade has built-in APIs for communication among platform-specific ontologies. However, these platform-specific ontologies cannot be shared over the Semantic Web. Jade has to import a third-party package, Bean Generator, to achieve adaptivity. Cougaar adopts Java-based message representation to maximize performance. Cougaar defines message content by pluggable components, but does not currently support high-level knowledge representation.

2. Agent interoperability over the agent platform

The FIPA interaction protocol forms a suite of protocol standards. Making agents compliant with FIPA facilitates interoperability over different agent platforms. Jfipa concerns about transferring message to agents, rather than interacting with agents. Thus, it does not support the FIPA interaction protocol suite. Jade supports most of the FIPA protocols, but unfortunately, provides only a protocol framework. Restated, users must generate agents' behavior as a finite states machine (FSM), and implement the FSM handling methods. Cougaar does not support the FIPA standard, but instead performs inter-agent communication on a blackboard based on the “Replys” mechanism. In other words, the users must implement “Relay.source” and “Relay.target” interfaces to exchange messages.

3. Integrativity of agents and the Semantic Web

Agents on the SemanticWeb act as coordinators between semantic web services. Integrating the Semantic Web with agents enables the agents to seamlessly coordinate semantic web services. Jfipa is a flexible framework for deploying agents to a new environment, but has no mechanism for integrating web services, and certainly not for semantic web services. Jade provides a web service integration gateway (WSIG) [7] to register web services. This gateway integrates agents with web services, but not with semantic web services. By contrast, Cougaar embeds Tomcat in its nodes (similar to agent platforms). A web service can be treated as a component of an agent. Thus, neither Jade nor Cougaar integrate agents with semantic web services.

3: ARCHITECTURE

Fig. 2 shows that this framework contains six components: 1) transport channel (that communicates with external transport channel), 2) reasoner, 3) agent mental model, 4) interaction protocol ontology, 5) OWL-S service resolver (which reads an OWL-S description), and 6) domain knowledge. Each is described in details next.

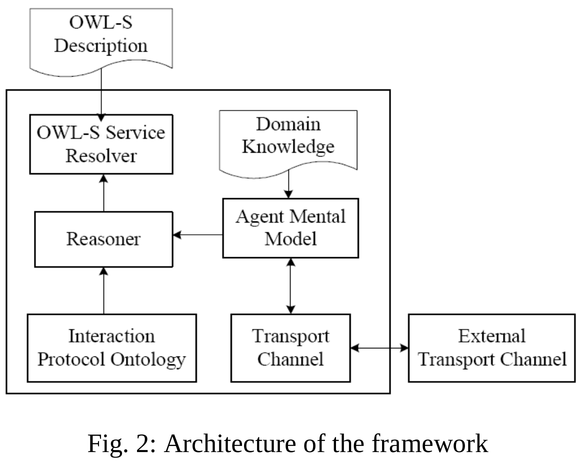

Fig. 2: Architecture of the framework

3.1: Transport Channel

A transport channel encodes and transports coordination messages. Agents running in heterogeneous environments (such as disparate operating systems) can carry communication states from one end of this channel to the other.

The transport channel has both a server and a client implementation. The server listens for incoming messages and the client dispatches outgoing messages. HTTP is used as the underlying transport protocol and XML as the encoding scheme. The Document Type Definition (DTD) of the coordination message is expressed as follows:

```
<?xml version="1.0"?>
<!--A Coordination-Message contains Communicative-Act,
Initiator, Participant, Service and Proposition-->

<!ELEMENT Coordination-Message (Communicative-Act,
Initiator, Participant+, Action?, Proposition*)>
<!ELEMENT Communicative-Act (#PCDATA)>
<!ELEMENT Initiator(#PCDATA)>
<!ELEMENT Participant (#PCDATA)>

<!-- An Action contains Service and Parameters -->
<!ELEMENT Action (Service, Parameters)>
<!ELEMENT Service (#PCDATA)>

<!-- Parameters contain Input and Output -->
<!ELEMENT Parameters (Input*,Output*)>
<!ELEMENT Input (#PCDATA)>
<!ELEMENT Output (#PCDATA)>
<!ATTLIST Input type CDATA #REQUIRED>
<!ATTLIST Output type CDATA #REQUIRED>

<!-- A Proposition contain subject, predicate and object-->
<!ELEMENT Proposition (subject, predicate, object)>
<!ELEMENT subject (#PCDATA)>
<!ELEMENT predicate (#PCDATA)>
<!ELEMENT object (#PCDATA)>
```

3.2: Reasoner

A reasoner, such as RACER [10], is compatible with description logics implementation group (DIG) [11] interface. It contains interaction protocol ontology and agent mental model, which form the knowledge base of this framework. In driving the interaction between agents, a reasoner accepts 1) a “tells” operator, which contains the communication states to update the knowledge base, and 2) an “asks” operator, which contains formal semantics [13] to query the knowledge base.

Below is an example about “tells” operator. An agent receives the communication states about someone who wants to coordinate a book buying service.

The “tells” operator is expressed as follows:

```
<tells
    uri="urn:protocol"
    xmlns="http://dl.kr.org/dig/2003/02/lang"
>
  <related>
```

```
    <individual name="urn:protocol#Mary"/>
    <ratom name="urn:protocol#coordinates"/>
    <individual name="urn:protocol#BookBuyingService"/>
  </related>
</tells>
```

In this example, a “tells” operator updates the knowledge base about “Mary coordinates the book buying service".

Next is an example about “asks” operator. John requests Mary to perform a book buying service. A part of formal semantics of a request communicative act is:

“B_{John} *Agent* (*Mary, BookBuyingService*)”

that means John believes Mary is the agent that has performed the book buying service before. In order to meet the formal semantic, agent John uses an “asks” operator to query the knowledge base to confirm whether he believes Mary has done so. The “asks” operator is expressed as follows:

```
<asks
    uri="http://dl.kr.org/dig/kb-51610"
    xmlns="http://dl.kr.org/dig/2003/02/lang"
>
  <toldValues id="q1">
    <individual name="urn:beliefbase#b1"/>
    <attribute name="urn:beliefbase#belief"/>
  </toldValues>
</asks>
```

In this example, an “asks” operator queries the knowledge base to make sure if John believes b1 (Mary coordinates a book buying service).

3.3: Agent Mental Model

An agent mental model defines the modality of belief [13], which denotes an agent's propositional attitude, for driving the communication. To preserve the communication states just mentioned, the initial communication states originating in the agent's belief base [14] are loaded into the mental model during framework initialization. The inferred states are immediately sent to the outside environment whenever new communication states are added to, or removed from, the mental model.

For example, an agent's mental model preserves a communication state expressed as the following web ontology language (OWL) code:

```
<rdf:RDF
<!--b1 states that Mary coordinates Book Buying Service-->
  <Proposition rdf:ID="b1">
    <subject>Mary</subject>
    <predicate>coordinates</predicate>
    <object>BookBuyingService</object>
    <belief>true</belief>
  </Proposition>

  <!--agent John believes b1-->
  <Agent rdf:ID="John">
    <believe rdf:resource="#b1">
  </Agent>
</rdf:RDF>
```

In this example, agent John believes b1, which states that “Mary coordinates Book Buying Service”. Restated, John believes that Mary coordinates the book buying service.

3.4: Interaction Protocol Ontology

The core of this framework is the OWL-based interaction protocol ontology (Fig. 3), which supplies a set of markup language constructs to describe the properties of inter-agent communication in an unambiguous, computer-interpretable form.

The interaction protocol ontology has three important classes, namely agent, communicative act and protocol.

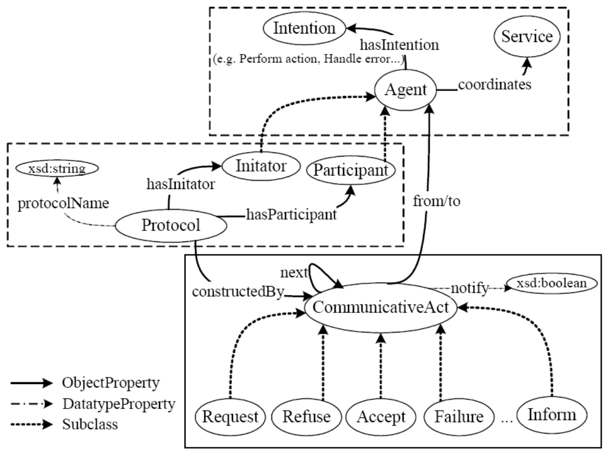

Fig. 3: Interaction protocol ontology

The agent describes:

1. The entities involved in coordinating semantic web services.
2. The service that enables agents to coordinate, specified by the property “coordinates”. Notably, the service is equivalent to the service in OWL-S.
3. The type of intention that the agent wishes to accomplish, specified by the property “hasIntention”. The intention may be to perform action, handle error, pass information, request information or negotiate agreement.

For example, an agent involved in coordinating a web service is expressed as follows:

```
<!--a Book Buying Service-->
<Service rdf:ID="BookBuyingService"/>

<!--Agent John intents to perform coordinating Book Buying
Service-->
<Agent rdf:ID="John">
  <coordinates rdf:resource="#BookBuyingService"/>
  <hasIntention rdf:resource="#PerformAction"/>
</Agent>
```

In this example, agent John has the intention to perform the action of coordinating the book buying service.

The communicative act describes:

1. The sender and receiver(s), specified by the properties “from” and “to”.
2. The next communicative act, specified by property “next”.
3. The communicative acts denoting performative utterance that can be derived. These acts may include, among others, “accept”, “inform”, “failure”, “request” and “refuse”. FIPA proposes 22 fundamental communicative acts [12] in agent communication.

For example, Fig. 4 shows a FIPA request interaction protocol consisting of three possible interaction sequences: (1) request-accept-inform, (2) request-accept-failure and (3) request-refuse. Notably, the abnormal act “NotUnderstood” is omitted.

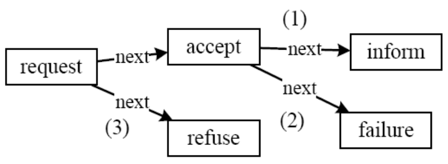

Fig. 4: The FIPA request interaction protocol and three possible interaction sequences

The sequences are expressed as follows:

```
<Initiator rdf:ID="John"/>
<Participant rdf:ID="Mary"/>

<!-- request-accept-inform -->
<Request rdf:ID="request">
   <from rdf:resource="#John"/>
   <to rdf:resource="#Mary"/>
   <next rdf:resource="#agree"/>
</Request>
<Agree rdf:ID="agree">
   <from rdf:resource="#Mary"/>
   <to rdf:resource="#John"/>
   <next rdf:resource="#inform"/>
</Agree>
<Inform rdf:ID="inform">
   <from rdf:resource="#Mary"/>
   <to rdf:resource="#John"/>
</Inform>

<!-- request-accept-failure -->
<Request rdf:ID="request">
   <from rdf:resource="#John"/>
   <to rdf:resource="#Mary"/>
   <next rdf:resource="#agree"/>
</Request>
<Agree rdf:ID="agree">
   <from rdf:resource="#Mary"/>
   <to rdf:resource="#John"/>
   <next rdf:resource="#failure"/>
</Agree>
<Failure rdf:ID="failure">
   <from rdf:resource="#Mary"/>
   <to rdf:resource="#John"/>
</Failure>

<!-- request-refuse -->
<Request rdf:ID="request">
   <from rdf:resource="#John"/>
   <to rdf:resource="#Mary"/>
   <next rdf:resource="#refuse"/>
</Request>
<Refuse rdf:ID="refuse">
   <from rdf:resource="#Mary"/>
   <to rdf:resource="#John"/>
</Refuse>
```

In this example, the “request” communicative act is sent from initiator “John” to participant “Mary”, while the rest of the acts are sent in the opposite direction.

The protocol describes:

1. The participants in the interaction, given in the two properties “hasInitator” and “hasParticipant”. The “hasInitiator” indicates a special participant who initiates the interaction.
2. The protocol adopted by the agent, described by the “protocolName” property.
3. The communicative acts specified by the “constructedBy” property, which construct the interaction protocol.

For example, an interaction protocol is expressed as follows:

```
<Protocol rdf:ID="protocol">
   <protocolName rdf:datatype="&xsd;#string"
   >request</protocolName>
   <hasInitiator rdf:resource="#John"/>
   <hasParticipant rdf:resource="#Mary"/>
   <constructedBy rdf:resource="#request"/>
   <constructedBy rdf:resource="#agree"/>
   <constructedBy rdf:resource="#refuse"/>
   <constructedBy rdf:resource="#inform"/>
   <constructedBy rdf:resource="#failure"/>
</Protocol>
```

In this example, an interaction protocol named “request” is constructed by “request”, “agree”, “refuse”, “inform”, and “failure” communicative acts, and includes initiator John and participant Mary.

The framework can be easily extended to accommodate other FIPA interaction protocols, simply by instantiating the interaction sequences and interaction protocol from the interaction protocol ontology (Fig. 3) as described above.

3.5: OWL-S Service Resolver

OWL-S service resolver extracts the service interface as an action of the coordination message (see Document Type Definition in section 3.1). An OWL-S description contains properties, namely serviceName, hasInput, hasOutput, hasPrecondition and hasResult [5], to present a service interface.

Figure 5 illustrates an OWL-S description, book buying service. This service has the “hasOutput” property, which specifies order status, and the “hasInput” properties to specify the book's title and the price.

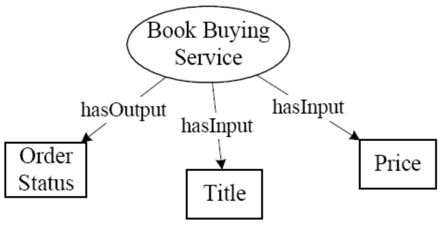

Fig.5: Book Buying Service

In this example, the action of coordination message is expressed as follows:

```
<Action>
   <Service>bookbuyingservice</Service>
   <Parameters>
      <Input type="http://www.swcf.org/bookbuying#Title"
      >Different Seasons</Input>
      <Input type="http://www.swcf.org/bookbuying#Price"
      >$9</Input>
      <Output
        type="http://www.swcf.org/bookbuying#OrderStatus"/>
   </Parameters>
</Action>
```

3.6: Domain Knowledge

Domain knowledge generally represents the particular meanings of terms that domain experts apply to a specific domain. This work defines domain knowledge as the type and the parameters of the services. Additionally, a service interface can be validated by domain knowledge when agents orchestrate a web service scenario.

Figure 6 illustrates the domain knowledge of a book buying service. According to the scenario in Fig. 5, the “service” sells “books”, and each book has “title”, “price” and order status “hasOrdered”. Additionally, the service has “members”, some of whom belong to the “VIP” group.

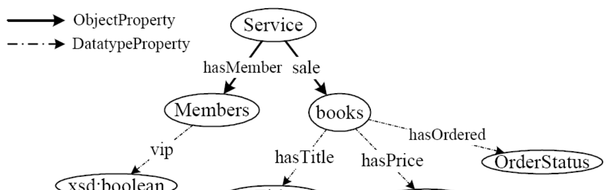

Fig. 6: Domain knowledge

4: AN EXAMPLE

John and Mary are customers of the Elite bookstore. She is a VIP member of the bookstore, while he is not. Both have agents to assist them with buying books in the bookstore (Fig. 5). The VIP members enjoy lower prices than others. Thus, John's agent holds a belief that *“John believes (a book) Different Seasons has a price of $9”* in its mental model, while Mary's agent believes *“Mary believes Different Seasons has a price of $6”*.

This example demonstrates that agents exploit the coordinative communication of this framework to buy a book “Different Seasons” for $6 in the book buying service.

With a limited budget of $6, John delegates his agent to buy “Different Seasons”. The agent inspects its mental model and knows that he cannot perform this task, since this violates its belief that *“Different Seasons has a price of $9”*. The agent asks the reasoner, “who is the participant of book buying service coordination?”, and the reasoner responds, *“Mary is the participant of book buying service coordination.”*

The agent then adds this proposition to its mental model as a belief. The agent again asks the reasoner which communicative act to apply to express its performative utterance. The reasoner replies, “Request”.

The coordination message for the above is:

```
<Coordination-Message>
   <Communicative-Act>Request</Communicative-Act>
   <Initiator>John</Initiator>
   <Participant>Mary</Participant>
   <Action>
      <Service>bookbuyingservice</Service>
      <Parameters>
      <Input type="http://www.swcf.org/bookbuying#Title"
      >Different Seasons</Input>
      <Input type="http://www.swcf.org/bookbuying#Price"
      >$9</Input>
      <Output
        type="http://www.swcf.org/bookbuying#OrderStatus"/>
   </Parameters>
   </Action>
</Coordination-Message>
```

The transport channel then transfers this coordination message to Mary's agent, which receives the message and asks the reasoner, “Is John a participant of coordination?” The reasoner replies, *“John is the participant of book buying service coordination.”* Then, Mary's agent adds this proposition to its mental model as a belief, and asks the reasoner which communicative act to apply. The reasoner replies with “Agree” and “Refuse”.

Mary's agent inspects its mental model to determine which communicative act to choose. Because it has a belief that *“Different Seasons has a price of $6”*, it believes Mary can help John with this task, and therefore chooses “Agree”.

The coordination message for the above is:

```
<Coordination-Message>
   <Communicative-Act>Agree</Communicative-Act>
   <Initiator>John</Initiator>
   <Participant>Mary</Participant>
   <Action>
      <Service>bookbuyingservice</Service>
      <Parameters>
      <Input type="http://www.swcf.org/bookbuying#Title"
      >Different Seasons</Input>
```

```
    <Input type="http://www.swcf.org/bookbuying#Price"
    >$9</Input>
    <Output
    type="http://www.swcf.org/bookbuying#OrderStatus"/>
  </Parameters>
  </Action>

  <Proposition>
   <subject>Different Seasons</subject>
   <predicate>http://www.swcf.org/bookbuying#hasPrice</p
redicate>
   <object>$6</object>
  </Proposition>
</Coordination-Message>
```

The transport channel then transfers this message to John's agent, which learns that the condition of the agreement (*"Different Seasons has a price of $6"*) meets John's budget.

Meanwhile, Mary's agent performs John's request of invoking book buying service to order a book "Different Seasons" for $6, and the book buying service replies the message *"Different Seasons has ordered"* to Mary's agent, which adds it to the mental model as a belief.

After performing this service, Mary's agent asks the reasoner which communicative act to apply. The reasoner replies with "Inform" and "Failure". Mary's agent inspects its mental model and believes that *"Different Seasons has been ordered"*, and thus chooses "Inform".

The coordination message for the above is:

```
<Coordination-Message>
  <Communicative-Act>Inform</Communicative-Act>
  <Initiator>John</Initiator>
  <Participant>Mary</Participant>
  <Proposition>
    <subject>Different Seasons</subject>
    <predicate>http://www.swcf.org/bookbuying#hasOrdere
d</predicate>
    <object>true</object>
  </Proposition>
</Coordination-Message>
```

Then, the transport channel transfers the message to John's agent, and the coordinative communication is completed.

5: CONCLUSIONS

This framework facilitates inter-agent communication and web service coordination over the Semantic Web. It has the following advantages:

1. All components of the framework are specified in web ontology language (OWL), a recommended standard for the Semantic Web. Therefore, this framework serves as a semantic web infrastructure, facilitating inter-agent communication over the Semantic Web.
2. By conforming to the same FIPA interaction protocols, agents in this framework serve as articulated communicators to orchestrate scenarios of semantic web services. Consequently, this framework coordinates semantic web services in a standardized manner.

Hence, this framework is envisioned to realize automated web service discovery and composition over the Semantic Web in the near future.

ACKNOWLEDGEMENTS

The authors would like to thank the National Science Council in Taiwan under Grant NSC94-2213-E-008-029, Department of Industrial Technology, Ministry of Economic Affairs in Taiwan, under Grant 95-EC-17-A-02-S1-029 and the Ministry of Education in Taiwan under Grant EX91-E-FA06-4-4.

REFERENCES

[1] D. Martin et al., OWL-S: Semantic markup for web services, DARPA, 2004, available at http://www.daml.org/services/owl-s/1.1/overview/

[2] J. Hendler, Agents and the Semantic Web, IEEE Intelligent Systems, vol. 16, no. 2, Mar. /Apr. 2001, pp. 30-37.

[3] Tim Berners-Lee, James Hendler and Ora Lassila, The Semantic Web, *Scientific American*, May 2001.

[4] Ethan Cerami, *Web Services Essentials*, O'Reilly Press, Feb. 2002, pp. 102-133.

[5] S. McIlraith, T.C. Son, and H. Zeng, Semantic Web Services, IEEE Intelligent Systems Special Issue on the Semantic Web, March/April, 2001, pp. 46-53.

[6] Amund Tveit, jfipa- An Architecture for Agent-based Grid Computing, In Proceedings of AISB'02 Convention, Symposium on AI and Grid Computing, London, United Kingdom, Apr. 2002.

[7] D. Greenwood, J. Nagy, M. Calisti, Semantic Enhancement of a Web Service Integration Gateway, Service Oriented Computing and Agent Based Engineering (SOCABE) Workshop, Netherlands, 2005.

[8] Aaron Helsinger, Michael Thome and Todd Wright, Cougaar: A Scalable, Distributed Multi-Agent Architecture, Systems, Man and Cybernetics, 2004 IEEE International Conference on Volume 2, Oct. 2004, pp. 10-13.

[9] G. Antoniou and F. van Harmelen, *A Semantic Web Primer*, The MIT Press, 2004, pp.10-12.

[10] M. Wessel and R. Möller, A High Performance Semantic Web Query Answering Engine, Proc. of International Workshop on Description Logics, 2005.

[11] S. Bechhofer, R. Möller, and P. Crowther, The DIG Description Logic Interface, In Proc. of International Workshop on Description Logics (DL2003), Rome, Italy, 2003.

[12] Foundation for Intelligent Physical Agents, FIPA Communicative Act Library Specification, Dec. 2002.

[13] H. Dekker, Possible Worlds, Belief, and Modal Logic: a Tutorial, Oct. 2004.

[14] Chang-Hyun Jo, Guobin Chen and James Choi, A new approach to the BDI agent-based modeling, Proceedings of the ACM symposium on Applied computing, Mar. 2004, pp. 1541-1545.