Implementation of Ontology Mapping for Computational Agents

ROMAN NERUDA
Institute of Computer Science
Academy of Sciences of the Czech Republic
P.O.Box 5, 18207 Prague, Czech Republic
roman@cs.cas.cz

Abstract: This paper describes ontological description of computational agents, their properties and abilities. The goal of the work is to allow for autonomous behavior and semi-automatic composition of agents within a multi-agent system. The system has to be create foundation for the interchangeability of computational components, and emergence of new models. This paper focuses on ways of representing agents and systems in standard formalisms, such as description logics, OWL, and Jade.

Key–Words: Multi-agent systems, Ontology, Computational intelligence.

1 Introduction

There is a lot of research in how to use formal logics to model ontologies. The goal of this research is to find logics that are both expressive enough to describe ontological concepts, and weak enough to allow efficient formal reasoning about ontologies. The most natural approach to formalize ontologies is the use of First Order Predicate Logics (FOL). This approach is used by well known ontology description languages like Ontolingua [5] and KIF [8]. The disadvantage of FOL-based languages is the expressive power of FOL. FOL is undecidable [3], and there are no efficient reasoning procedures. Nowadays, the de facto standard for ontology description language for formal reasoning is the family of description logics. Description logics are equivalent to subsets of first order logic restricted to predicates of arity one and two [2]. Autonomous agents are small self-contained programs that can solve simple problems in a well-defined domain [13]. In order to solve complex problems, agents have to collaborate, forming Multi-Agent Systems (MAS). A key issue in MAS research is how to generate MAS configurations that solve a given problem [4]. In most Systems, an intelligent (human) user is required to set up the system configuration. Developing algorithms for automatic configuration of Multi-Agent Systems is a major challenge for AI research. In this paper, we introduce a logical reasoning component for MAS. With this component, system configurations can be created automatically and semi-automatically. The logical description of MAS allows for interaction with ontology based distributed knowledge systems like the Semantic Web [9].

Description logics is used as the underlying framework for the Semantic Web, a project of the Internet standardization body W3C. The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers to deal with that information in a formal way. [1]. The Knowledge Grid project [18], [17] builds on top of the Semantic Web to create an intelligent environment allowing agents (both software and human) to share and manage knowledge. The objectives of the Knowledge Web are to support of team-work, problem-solving and decision making. Description logics is also the main topic of interest in other projects dealing with the standardization of inter-agent communications.

2 Computational Agents

An *agent* is an entity that has some form of perception of its environment, can act, and can communicate with other agents. It has specific skills and tries to achieve goals. A *Multi-Agent System (MAS)* is an assemble of interacting agents in a common environment [6].

In order to use automatic reasoning on a MAS, the MAS must be described in formal logics. We define a formal description for the static characteristics of the agents, and their communication channels. We do not model dynamic aspects of the system yet. We assume the Jade [10] type of agents communicating by FIPA [7] compliant messages. A conversation between two agents usually consists of a number of messages. In order to abstract from the actual messages, we subsume all messages of the similar type of conversation under a *message type*.

Concepts	
mas(C)	C is a Multi-Agent System
class(C)	C is the name of an agent class
gate(C)	C is a gate
m_type(C)	C is a message type
Roles	
type(X,Y)	Class X is of type Y
has_gate(X,Y)	Class X has gate Y
gate_type(X,Y)	Gate X accepts messages of type Y
interface(X,Y)	Class X understands mess. of type Y
instance(X,Y)	Agent X is an instance of class Y
has_agent(X,Y)	Agent Y is part of MAS X

Table 1: Concepts and roles used to describe MAS.

A *message type* identifies a category of messages that can be send to an agent in order to fulfill a specific task. We refer to message types by unique identifiers. The set of message types understood by an agent is called its *interface*. For outgoing messages, each link of an agent is associated with a message type. Via this link, only messages of the given type are sent. We call a link with its associated message type a *gate*.

Now it is easy to define if two agents can be connected: Agent A can be connected to agent B via gate G if the message type of G is in the list of interfaces of agent B. Note that one output gate sends messages of one type only, whereas one agent can receive different types of messages. This is a very natural concept: When an agent sends a message to some other agent via a gate, it assigns a specific role to the other agent, e.g. being a supplier of training data. On the receiving side, the receiving agent usually should understand a number of different types of messages, because it may have different roles for different agents. A *connection* is described by a triple consisting of a sending agent, the sending agent's gate, and a receiving agent.

Next we define *agents* and *agent classes*. Agents are created by generating instances of classes. An agent class is defined by an interface, a set of message types, a set of gates, and a set of types. An agent is an instance of an agent class. It is defined by its name and its class.

Multi-Agent Systems are assemblies of agents. For now, only static aspects of agents are modeled. Therefore, a Multi-Agent System can be described by three elements: The set of agents in the MAS, the connections between these agents, and the characteristics of the MAS (constraints). *Multi-Agent Systems (MAS)* consist of a set of agents, a set of connections between the agents, and the characteristics of the MAS.

3 Description logics and Lisp

Description logics know concepts (unary predicates) and roles (binary predicates). In order to describe agents and Multi-Agent Systems in description logics, the above definitions are mapped onto description logic concepts and roles as shown in table 1.

Probably the most natural implementation of the described framework of agents and their communication-related properties is by means of a description logics formalism. We have chosen the Lisp-like syntax of the RACER ontological reasoning engine [11], because the transcription from the above definitions is quite straightforward. The knowledge base description contains TBox and ABox declarations. In our case, the TBox consists of the (static) description of classes and properties, while the ABox represents the dynamic information about the current state of the running system. TBox declarations are known in advance, while ABox is populated by means of communication with the directory services (so-called yellow pages) agents.

The example of concrete concepts and roles implemented in the RACER Lisp syntax is provided in Figure 2. For the sake of simplicity, only partial information is included. The complete description is included in [12], and the overall graphical scheme is presented in Figure 1.

```
(signature :atomic-concepts (classInBang
     iAgentStdIface
     agentLifeManagement
     igToYellowPages
     yellowPageRequest
     Father
     igData
     DataSource
     DataSourceConsumer
     igCommonCompControl
     igIterativeCompControl
     Computation
     IterativeComputation
     NonIterativeComputation
     igFunction
     Function
     igStoreModel
     igQueryModel
     queryModel
     Model
     Approximator
     Classifier
     TaskManager
     aDecisionTree
     NeuralNetwork
     RBFNetwork
     MultiLayerPerceptron
     aYellowPages
...
     )
   :roles (interface
   gate
   messagetype
   has
   hide
       instanceof
   )
   )
```

Figure 2: Overview of concepts and roles in the RACER Lisp-like formalism.

The detailed case of several concrete agent classes in this syntax is shown in figure 3. One can see a part of hierarchy of agent classes consisting of a *Neural network* derived from a *Model* (via *Approximator*). There are also siblings of *Approximator* (*Classifier*) and two types of different ancestors of *Neural Network*: *Multilayer Perceptron* and *RBF Network*. Most of these classes not only inherit their properties, but add further gates and interfaces.

```
...
;;Models
(implies Model (and Function
DataSourceConsumer
(some interface igStoreModel)
(some interface igQueryModel)
(all interface (or igStoreModel igQueryModel))))

(implies Approximator Model)

(implies Classifier Model)
;;Neural Networks
(implies NeuralNetwork Approximator)

;;RBF Network
(implies RBFNetwork (and NeuralNetwork
IterativeComputation
classInBang
SimpleTaskManager
Father
(some gate igSolveRepresentatives)
(some hide igCommonCompControl)
(all hide igCommonCompControl)
(some gate igSolveLinEqSystem)
(all gate (or igSolveRepresentatives igSolveLinEqSystem))
(some interface igRunNetworkDemo)
(all interface igRunNetworkDemo)))

;;Multilayer Perceptron
(implies MultiLayerPerceptron (and NeuralNetwork
IterativeComputation
classInBang
Father))
```

Figure 3: Several agent classes in native Lisp format of the RACER reasoning system.

4 OWL representation

The *Web Ontology Language (OWL)* from the World Wide Web Consortium [16] is the most recent member of the standard ontology languages family. OWL makes it possible for concepts to be defined as well as described. Complex concepts can therefore be easily built from simpler ones. Furthermore, the logical model allows the use of a reasoner which can check whether or not all of the statements and definitions in the ontology are consistent. The reasoner can help to maintain the hierarchy correctly which is useful when dealing with cases where classes can have more than one parent [14]. Syntactically, the OWL has a form of XML.

There are three types of OWL distinguished by their expressive power. The least expressive is the OWL-Lite suitable for expressing quite simple relations only. The OWL-DL is much more powerful since it is based on the Description Logics. The third one, OWL-Full, is the most powerful in terms of expressiveness, but it is not suitable for machine reasoning because the ontologies can be neither complete nor decidable. Naturally, the OWL-DL is the most suitable candidate for our approach.

The following agent class *Model* derived from *Function*, enhancing it with particular interfaces, is described like this:

$$\begin{aligned}&\text{MODEL} \subseteq \text{FUNCTION} \cap \text{DATASOURCECONSUMER} \cap \\ &(\text{INTERFACE some IGSTOREMODEL}) \cap \\ &(\text{INTERFACE some IGQUERYMODEL}) \cap \\ &(\text{INTERFACE only } (\text{IGSTOREMODEL} \cup \text{IGQUERYMODEL}))\end{aligned}$$

The OWL implementation of this class is in Figure 4. The subclass is defined by the `rdfs:subClassOf` keyword, constraints are in the `owl:Restriction` clause. Another keywords for relations are `owl:intersectionOf`, `owl:unionOf`, `owl:someValuesFrom`, and `owl:allValuesFrom`. Figure 5 represents the OWL implementation of the *RBF Network* class described also in the previous section.

```
<owl:Class rdf:about="http://MODEL">
 <rdfs:subClassOf>
  <owl:Class>
   <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="http://FUNCTION"/>
    <owl:Class rdf:about="http://DATASOURCECONSUMER"/>
    <owl:Restriction>
     <owl:someValuesFrom>
      <owl:Class rdf:about="http://IGSTOREMODEL"/>
     </owl:someValuesFrom>
     <owl:onProperty rdf:resource="http://INTERFACE"/>
    </owl:Restriction>
    <owl:Restriction>
     <owl:someValuesFrom>
      <owl:Class rdf:about="http://IGQUERYMODEL"/>
     </owl:someValuesFrom>
      <owl:onProperty rdf:resource="http://INTERFACE"/>
    </owl:Restriction>
    <owl:Restriction>
     <owl:onProperty rdf:resource="http://INTERFACE"/>
      <owl:allValuesFrom>
       <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
         <owl:Class rdf:about="http://IGSTOREMODEL"/>
         <owl:Class rdf:about="http://IGQUERYMODEL"/>
        </owl:unionOf>
       </owl:Class>
      </owl:allValuesFrom>
    </owl:Restriction>
   </owl:intersectionOf>
  </owl:Class>
 </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:about="http://APPROXIMATOR">
  <rdfs:subClassOf rdf:resource="http://MODEL"/>
</owl:Class>
<owl:Class rdf:about="http://NEURALNETWORK">
  <rdfs:subClassOf rdf:resource="http://APPROXIMATOR"/>
</owl:Class>
```

Figure 4: Three agent classes in the OWL-DL.

5 Integration with Jade/FIPA

So far, we are able to represent the agents, their properties and communication constraints in the OWL-DL and reason about them in a formal way with the help

```
<owl:Class rdf:about="http://RBFNETWORKAI">
 <rdfs:subClassOf>
  <owl:Class>
   <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="http://NEURALNETWORK"/>
    <owl:Class rdf:about="http://ITERATIVECOMPUTATION"/>
    <owl:Class rdf:about="http://CLASSINBANG"/>
    <owl:Class rdf:about="http://SIMPLETASKMANAGER"/>
    <owl:Class rdf:about="http://FATHER"/>
    <owl:Restriction>
     <owl:someValuesFrom>
      <owl:Class rdf:about="http://IGSOLVEREPRESENTATIVES"/>
     </owl:someValuesFrom>
     <owl:onProperty rdf:resource="http://GATE"/>
    </owl:Restriction>
    <owl:Restriction>
     <owl:onProperty rdf:resource="http://HIDE"/>
     <owl:someValuesFrom>
      <owl:Class rdf:about="http://IGCOMMONCOMPCONTROL"/>
      </owl:someValuesFrom>
    </owl:Restriction>
    <owl:Restriction>
     <owl:onProperty rdf:resource="http://HIDE"/>
      <owl:allValuesFrom>
       <owl:Class rdf:about="http://IGCOMMONCOMPCONTROL"/>
       </owl:allValuesFrom>
    </owl:Restriction>
    <owl:Restriction>
     <owl:onProperty rdf:resource="http://GATE"/>
     <owl:someValuesFrom>
       <owl:Class rdf:about="http://IGSOLVELINEQSYSTEM"/>
     </owl:someValuesFrom>
    </owl:Restriction>
    <owl:Restriction>
     <owl:onProperty rdf:resource="http://GATE"/>
     <owl:allValuesFrom>
      <owl:Class>
       <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="http://IGSOLVEREPRESENTATIVES"/>
        <owl:Class rdf:about="http://IGSOLVELINEQSYSTEM"/>
       </owl:unionOf>
      </owl:Class>
     </owl:allValuesFrom>
    </owl:Restriction>
    <owl:Restriction>
     <owl:onProperty rdf:resource="http://INTERFACE"/>
      <owl:someValuesFrom>
       <owl:Class rdf:about="http://IGRUNNETWORKDEMO"/>
      </owl:someValuesFrom>
    </owl:Restriction>
    <owl:Restriction>
     <owl:onProperty rdf:resource="http://INTERFACE"/>
     <owl:allValuesFrom rdf:resource="http://IGRUNNETWORKDEMO"/>
    </owl:Restriction>
   </owl:intersectionOf>
  </owl:Class>
 </rdfs:subClassOf>
</owl:Class>
```

Figure 5: RBF Network agent class in the OWL-DL.

on any compatible logical reasoner, such as RACER. This allows us to propose and verify scenarios of communication and ways of connecting individual agents into multi agent systems. However, in order to use this ontology within the agent communication messages, we have to modify it to comply with the FIPA-ACL definitions and platform-specific requirements. In our case, to make full use of automated application ontology and semantics handling in the JADE agent environment [10], one should modify the OWL-DL ontology developed above. In order to perform the proper semantic checks on a given content expression the platform has necessary to classify all possible elements that can appear within a valid sentence message according to their generic semantic characteristics. FIPA-ACL specification requires the content of each to have a proper semantics according to the performative of the ACLMessage. In particular, every ontology should be represented by a Java class derived from the JADE class `jade.content.onto.Ontology`.

There are additional classes `Concept`, `Predicate`, `AID` and `AgentAction` that serve as basic classes to inherit from in case of deriving ontological elements. The Figures 6 and 7 present the hierarchy of classes including *Model*, *Approximator*, *Classifier*, and *Neural Network*.

```
<?xml version="1.0"?>
<rdf:RDF
 xmlns:p1="http://#"
 xmlns="http://Ontology1159685804.owl#"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
 xmlns:owl="http://www.w3.org/2002/07/owl#"
 xml:base="http://Ontology1159685804.owl">
 <owl:Ontology rdf:about="">
  <owl:imports rdf:resource="http://"/>
 </owl:Ontology>
 <p1:JADE-CLASS rdf:ID="approximator">
  <rdfs:subClassOf>
   <p1:JADE-CLASS rdf:ID="model">
    <rdfs:subClassOf>
     <p1:JADE-CLASS rdf:ID="function">
      <rdfs:subClassOf rdf:resource="http://#AID"/>
     </p1:JADE-CLASS>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
     <p1:JADE-CLASS rdf:ID="iquerymodel">
      <rdfs:subClassOf>
       <p1:JADE-CLASS rdf:ID="interface">
        <rdfs:subClassOf rdf:resource="http://#Concept"/>
       </p1:JADE-CLASS>
      </rdfs:subClassOf>
     </p1:JADE-CLASS>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
     <p1:JADE-CLASS rdf:ID="istoremodel">
      <rdfs:subClassOf rdf:resource="#interface"/>
     </p1:JADE-CLASS>
    </rdfs:subClassOf>
   </p1:JADE-CLASS>
  </rdfs:subClassOf>
 </p1:JADE-CLASS>
 <p1:JADE-CLASS rdf:ID="classifier">
  <rdfs:subClassOf rdf:resource="#model"/>
 </p1:JADE-CLASS>
 <p1:JADE-CLASS rdf:ID="neuralnetwork">
  <rdfs:subClassOf rdf:resource="#approximator"/>
 </p1:JADE-CLASS>
 <p1:JADE-CLASS rdf:ID="gate">
  <rdfs:subClassOf rdf:resource="http://#Concept"/>
 </p1:JADE-CLASS>
</rdf:RDF>
```

Figure 7: JADE/FIPA compliant definition of hierarchy of several agent classes containing concrete interfaces.

Since the process from OWL-DL description to JADE/FIPA formalism is quite straightforward, it can be provided automatically by means of various tools. In our approach we have used the Protege [15] software with several plugins enabling automatic generation of Java code from the OWL description.

6 Conclusion

We have shown how formal logics can be used to describe computational agents. We presented a logical formalism for the description of agents and MAS. In this, we have started from ideal Description Log-

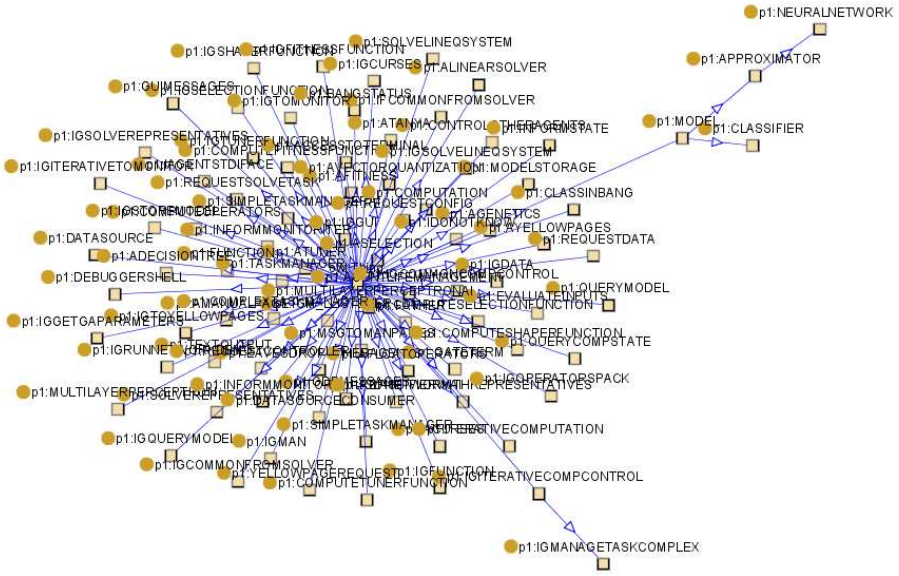

Figure 1: Scheme of the agent classes hierarchy.

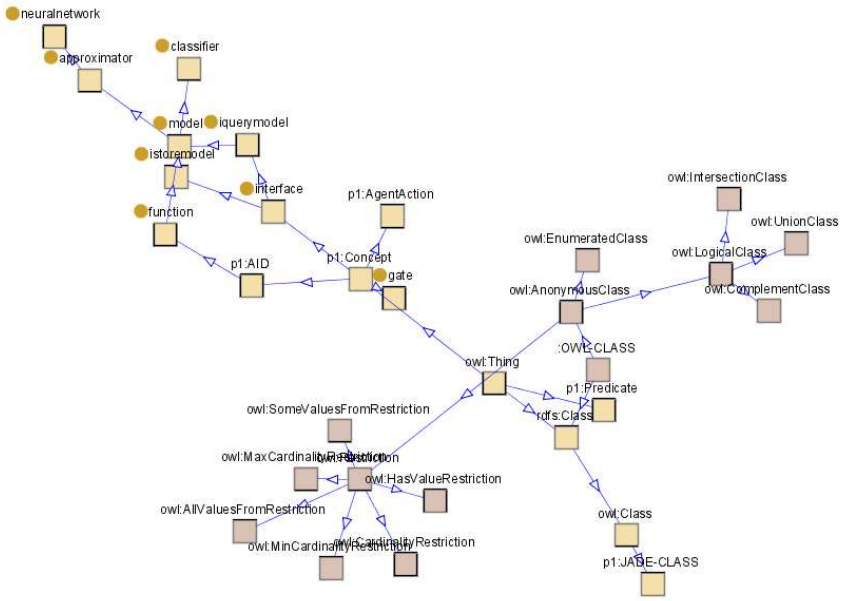

Figure 6: Graph of a JADE/FIPA compliant definition of hierarchy of several agent classes containing concrete interfaces.

ics and proceeded toward practical implementations by means of RACER Lisp format, OWL-DL, and JADE/FIPA-ACL compatible syntax. The system we implemented allows the practical application of these technologies. We have demonstrated how this approach works in practice within the hybrid computational environment.

So far, we only describe static aspects of MAS. Further research will be put in the development of formal descriptions of dynamic aspects of MAS. In particular, this means to work with ontological description of tasks and to gather knowledge about computational agents performance. Currently, there is a BDI-based mechanism that supports decisions of a computational agent based on its previous experience. This will blend smoothly with our approach, which in turn allows to provide more suitable MAS solutions. In particular, if there are more agents satisfying the constrains, we will be able to sort them according to their past performance in the required context. Thus, better partners for an agent can be supplied. Further in the future we plan to employ proactive mechanisms for an agent (again BDI-based), which will be allowed to improve its knowledge in its free time, such as trying to solve benchmark tasks and recording the results.

The hybrid character of the system, with both a logical component and soft computing agents, also makes it interesting to combine these two approaches in one reasoning component. In order to automatically come up with feasible hybrid solutions for specific problems, we plan to combine two orthogonal approaches: a soft computing evolutionary algorithm with a formal ontology-based model.

Acknowledgments

This work has been supported by the the project 1ET100300419 of the Program Information Society (of the Thematic Program II of the National Research Program of the Czech Republic) "Intelligent Models, Algorithms, Methods and Tools for the Semantic Web Realization".

References:

[1] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, May 2001.

[2] Alexander Borgida. On the relative expressiveness of description logics and predicate logics. *Artificial Intelligence*, 82(1–2):353–367, 1996.

[3] M. Davis, editor. *The Undecidable—Basic Papers on Undecidable Propositions, Unsolvable Problems and Computable Functions*. Raven Press, 1965.

[4] J. E. Doran, S. Franklin, N. R. Jennings, and T. J. Norman. On cooperation in multi-agent systems. *The Knowledge Engineering Review*, 12(3):309–314, 1997.

[5] A. Farquhar, R. Fikes, and J. Rice. Tools for assembling modular ontologies in ontolingua. Technical report, Stanford Knowledge Systems Laboratory, 1997.

[6] Jacques Ferber. *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Harlow: Addison Wesley Longman, 1999.

[7] Foundation for intelligent physical agents, 2006. http://www.fipa.org.

[8] Michael R. Genesreth and Richard E. Fikes. Knowledge interchange format, version 2.2. Technical report, Computer Science Department, Stanford University, 1992.

[9] J. Hendler. Agents and the semantic web. *IEEE Intelligent Systems*, 16(2):30–37, 2001.

[10] JADE: Java agents development environment, 2006. http://jade.tilab.com.

[11] Ralf Moeller. Racer ontology reasonner, 2006. http://www.sts.tu-harburg.de/ r.f.moeller/racer/.

[12] R. Neruda et al. Bang web documentation, 2006. http://bang.sf.org.

[13] H. S. Nwana. Software agents: An overview. *Knowledge Engineering Review*, 11(2):205–244, 1995.

[14] OWL W3C ontology language overview, 2004. http://www.w3.org/TR/owl-features/.

[15] Protege ontology tool, 2006. http://protege.stanford.edu.

[16] World wide web consortium, 2006. http://www.w3.org/.

[17] Hai Zhuge. Semantic space grid: Model, method and platform. *Concurrency and Computation:Practice and Experience*, 2004. in press.

[18] Hai Zhuge. Semantics, resource and grid. *Future Generation Computer Systems*, 20(1):1–5, 2004.