Applying Semantic Capability Matching into Directory Service Structures of Multi Agent Systems

Geylani Kardas[1], Özgür Gümüs[2], Oğuz Dikenelli[2]

[1] Ege University, International Computer Institute,
35100 Bornova, İzmir, Turkey
geylani@bornova.ege.edu.tr

[2] Ege University, Department of Computer Engineering,
35100 Bornova, Izmir, Turkey
{gumus,oguzd}@staff.ege.edu.tr

Abstact. In this paper, we introduce a semantically enriched capability matching model for agent services. Our vision is to integrate both agent and semantic web services and provide the interoperability of agents under the semantic web extension. In multi agent system architectures, there is a specific agent or service called directory facilitator which is responsible to keep knowledge about the services given by the agents within the system. Other agents query in directory facilitator to identify agents that provide the required services. Hence, automated service discovery in multi agent systems is a critical issue. Here, we propose a matching engine architecture in which capabilities of agent services are handled semantically and it replies agent service requests with *most suitable* service advertisements. The paper includes formal basics and design details of this engine and also discusses its implementation with a proper case study.

1 Introduction

Semantic Web [2] evolution has doubtlessly brought a new vision into agent research. This *Second Generation Web* aims to improve WWW (World Wide Web) such that web page contents are interpreted by using ontologies. It is apparent that the interpretation in question will be realized by autonomous computational entities –so agents- to handle semantic content on behalf of their human users.

Semantic Web vision obviously effects the current mainstream research directions in agent technologies [16] especially considering agent modeling, multi agent system (MAS) architectures and MAS methodologies.

Various studies on semantic web and agent systems integration have already produced new artifacts like new agent systems, agent development frameworks, etc. For example, in [3] a smart meeting room system is introduced in which agents provide relevant services and information to the meeting participants based on their contexts. Another system offers access to information about activities such as talks or seminars [6]. This system uses DAML+OIL (DARPA Agent Markup Language +

Ontology Inference Layer) [4] for knowledge representation and lets agents to retrieve and manipulate information stored in a proper knowledge base.

Those above mentioned systems use multi agent development frameworks like Jackal [5] and Jade [1] in their underlying infrastructure. However those frameworks do not have a built-in support for semantic web and system developers encounter difficulties in supporting basic semantic web functionalities such as automated discovery and dynamic invocation of agent services.

We believe that semantic web enabled multi agent systems can only be developed by using frameworks which will internally support basic semantic web functionalities and facilitate integration of semantic web and agents. We introduced such a framework in [7] and called it *SEAGENT*. SEAGENT is FIPA-compatible [8] and it looks like other existing agent development frameworks such as DECAF [10] and JADE [1]. However it includes several built-in features that the existing agent frameworks and platforms do not have.

Agents created using SEAGENT can handle their internal knowledge base using semantic web standards. Directory facilitator of SEAGENT is implemented in a way that it supports semantic matching of the agent capabilities.

In this paper, we introduce design basics and give implementation details of SEAGENT's semantic matching engine which will improve directory services in multi agent systems. We believe that agent platforms developed with SEAGENT will have more powerful yellow page services and easily realize semantic capability matching on agent services which will be a must in future's semantically enriched environments.

The rest of the paper is organized as follows: Section 2 gives our motivation and idea behind semantically enriched MAS directory services. This section also exposes our capability matching model and formal basics of the developed semantic matching engine. Section 3 explains the internal architecture and software design of the matching engine. A case study on proposed semantic capability matching is discussed in Section 4. Section 5 includes the conclusion and future work.

2 Semantic Matching for MAS Directory Services

Members of a multi agent platform need services offered by other members during execution of their plans and doing the jobs on behalf of their users. So, they inevitably look for those requested services in a predefined service registry. No matter it is FIPA-compliant [8] or not, a MAS owns one or more registries which provides yellow page services for system's agents to look for proper agent services. Of course registries mentioned above are not simple structures and mostly implemented as directory services and served by some platform specific agents. For example there is a mandatory agent called *directory facilitator (DF)* in FIPA abstract architecture specification on which agent services are registered. When an agent looks for a specific agent service, it gathers supplier data (agent's name, address, etc.) of the service from the DF and then it begins to communicate with this service provider agent to complete its task.

Matchmaking could be defined as the process of verifying whether a capability specification "*matches*" the specification of a request (e.g. a task to be solved) [9]. Two specifications "match" if their specifications verify some matching relation, where the matching relation is defined according to some criteria (e.g. a capability being able to solve a task). This matching may consider semantic relation(s) between these two specifications (advertised and requested). Therefore, in case of agent service discovery, we should define semantic matching criteria of service capabilities and design registration mechanisms (directory services) of agent service specifications according to those criteria. That makes matching of requested and advertised services more efficient by not only taking into consideration of identical service matching: New capability matching will determine type and degree of relation between two services (requested and advertised) *semantically*.

Capability matching of services is not a new idea and several studies exist in literature, proposing algorithms especially for discovery of semantic web services. For example studies in [12] and [14] introduce a capability matching on semantic web services which are modeled in DAML-S (DAML Services) [15]. Service profiles of both requested and advertised services are processed and a match between these profiles is determined when the advertised service could be used in place of the requested service.

On the other hand, [9] aims extending matchmaking on MAS environments in order to maximize the reuse of capabilities and tasks over new domains. To achieve this goal, the use of a knowledge modeling framework as the basis of an agent capability description language is proposed.

However, according to our vision; new generation multi agent environments will be semantically enriched by all means of agent interactions. These environments will be open to semantic web services in addition to existing agent services. Depending of its kind and needs, a service may be implemented as stand-alone (semantic web services) or an autonomous structure (an agent) as traditionally serves it. An agent may use a semantic web service and/or a service provided by another agent during its plan execution. So, middle agents of such platforms should realize matching on all those kind of services. At this point of view; our proposal on capability matching of services differentiates from the others mentioned above. We also take this semantically improved MAS vision into automated composition and interaction of services as proposed in [7] but those issues are beyond the scope of this paper.

Before discussing capabilities of the proposed semantic matching engine, it is worth for giving the formal basics of the match process in question:

Let;

O be an ontology defined using an ontology language (such as OWL [13]) and

C_1 and C_2 are classes (concepts) defined in O.

Definition 1: For all $C_X \in O$; C_1 is a direct subclass of C_2 if:

$(C_X \neq C_1) \wedge$

$(C_X \neq C_2) \wedge$

$((C_X \supset C_1) \wedge (C_X \subset C_2)) \Rightarrow C_X = \varnothing$

Definition 2: C_1 is a distant subclass of C_2 if:

$(C_2 \supset C_1)$ and C_1 is not direct subclass of C_2

Definition 3: *DoM(C_1, C_2)* is a degree of match function which determines semantic match degree between concepts C_1 and C_2 such that:

DoM(C_1 , C_2) = exact if C_1 is a direct subclass of C_2 or $C_1 = C_2$
DoM(C_1 , C_2) = plug-in if C_1 is a distant subclass of C_2
DoM(C_1 , C_2) = subsumes if C_2 is a direct or distant subclass of C_1
DoM(C_1 , C_2) = fail otherwise

We have used those relation and function definitions to design and implement the capability matching engine which works on agent services. In addition to other properties, an agent's service has a type which is a concept (or class) predefined in a domain ontology. Matching engine of the agent platform takes the above defined relations into account and determines the *suitability* of the advertised agent services with the requested one.

3 Seagent Matching Engine

We designed and implemented a capability matching engine for multi agent platforms in which directory services are semantically enriched. This matching engine -called *Seagent Matching Engine*- receives service requests and matches "semantically right" agent services with given service requests.

The engine stores agent service definitions in a database. Actually this database is an ontology model of the agent services in which agent service ontology individuals are included. Each agent service that is registered to the directory facilitator is also represented in this ontology with an individual. The matching engine uses those individuals and compares them with given service requests semantically.

The fact that a reasoning mechanism is needed to find out conceptual relations between individuals, a simple reasoner called *Ontolog* is developed within the engine. In our implementations, Ontolog is used by the matching engines to determine how the given two ontology concepts are related to each other and obtain the degree of the relationship if it exists. To perform its operations, Ontolog uses domain ontologies which may be web or locally enabled.

We implemented above proposed matching engine as a software package which can be used by directory service providers of any multi agent system (FIPA-compliant or not) to enhance their yellow page services with semantic capability matching. Fig. 1 shows the object model of the developed engine software.

DescriptionDB class is the database component of the software in which agent service descriptions are stored in an ontology model. For example, by using OWL we prepare an Agent Management Ontology in proper to FIPA Agent Management Specification (AMS) [8] so that the database stores directory facilitator agent descriptions as individuals. Each individual also includes its service description with a service type. Service types come from a predefined service ontology so that the engine can perform capability matching on requested and advertised services. Sample service ontology for a tourism domain is given in Fig. 2.

Ontolog is the generic interface that represents the above mentioned primitive reasoner of the matching engine. We implemented this engine component's various implementations in our studies and those were formerly dealing with DAML ontologies. The new version called *OwlOntolog* has been designed to process on OWL ontologies. It parses OWL documents and finds ontology concept distances.

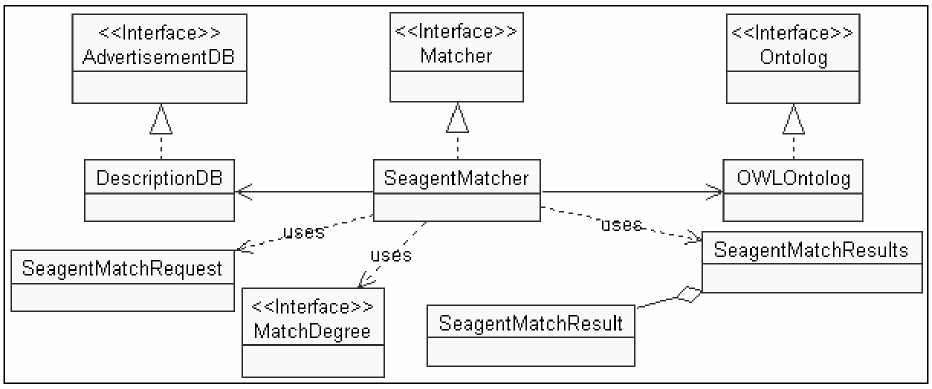

Fig. 1. Object model of the Seagent matching engine

Basically, OwlOntolog gives the superclass distance of the two ontology class with given URIs by determining “subClassOf” relations on the ontology. Consider the simple ontology class tree given in Fig. 2. Nodes of the ontology tree are service types of agents serving in a simple tourism domain. According to the ontology model, OwlOntolog finds superclass distances as -1, 0, 1 and 2 in (SeaTourism, HealthTourism), (SeaTourism, SeaTourism), (SeaTourism, SummerTourism), (SeaTourism, Tourism) ontology class pairs respectively. In case of a multiple inheritance, there will be different paths from a subclass to its superclass(es). At this condition OwlOntolog returns the shortest distance as a result of performing a depth-first search on the ontology tree.

Calculated ontology class distances are cached as instances in OwlOntolog to optimize performance. When the same distance query is received multiple times, they are all responded (excluding the first one) via ontolog’s cache.

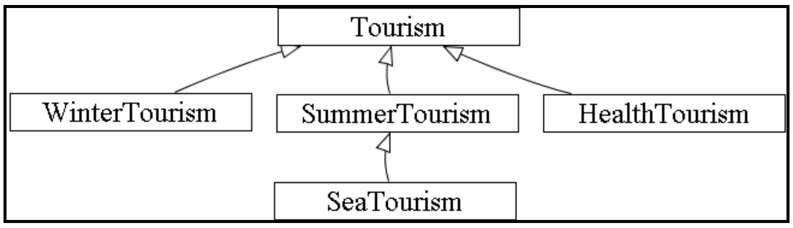

Fig. 2. A fragment of tourism ontology

SeagentMatcher class is the core and the most important component of the engine which manages and uses both reasoner and description database(s). MAS directory facilitators redirects platform agent’s service lookups to the SeagentMatcher. The SeagentMatcher uses its reasoner (OWLOntolog) and agent service description database to perform capability matching. Our design of the matcher gave us the flexibility of matching procedure such that it is completely independent of the number, type and ontology hierarchy of parameter(s) being used in the capability matching. For example a capability matching request may be only on agent service types and it includes simple semantic matching on services. On the other hand another request may not include a semantic match and it needs only the services supplied from a specific agent or agent group. More complicated agent service request involves both capability matching on service types and non-semantic request criteria like name of the supplier agent or service, agent's address, resolvers, etc.

The proposed engine meets all the requirements given above and enriches lookup services by providing matching for any level of match complexity. During its execution, it first performs an RDF (Resource Description Framework) query on individuals of the database model by RDQL (RDF Data Query Language) [11] filtering to find out proper individuals according to the given non-semantic parameters. Then the engine realizes a semantic capability matching on those filtered individuals and determines the result set of the match request. Such a filtering mechanism both supplies the management of complicated match requests and improves the performance by reducing number of database members to be processed by the reasoner. Of course the above pre-filtering will not be performed in a case of request in which only semantic match on database members is needed.

Each match request is encapsulated by a *SeagentMatchRequest* object which is given to matching engine to be processed on. A SeagentMatchRequest stores a match request in RDF triples. Sentences belong to semantic and non-semantic queries are given as form of RDF triples and those triples are stored in proper collections in the object. The request object also includes the desired match degree of the requester.

SeagentMatcher determines the semantic match degree between the requested and advertised service descriptions by calling appropriate OwlOntolog's method with service types of those descriptions. Calculated ontology class distance will be evaluated in the SeagentMatcher as follows:

```
If distance = 0 or distance = 1 then EXACT match is determined
If distance > 1 then PLUGIN match is determined
If distance < 0 then look for a reverse distance calculation
  (call finder with parameters in reverse order)
  If reverse_distance > 0 then SUBSUMES match is determined
  Else FAIL in match is determined
```

Each match result is encapsulated in a *SeagentMatchResult* including its degree of match. In software design, each match result implements a programming language specific "Comparable" interface to be easily and quickly sorted so agent service requester can retrieve service descriptions in order - from semantically most exact to least one.

Due to its proper API (Application Program Interface) design, use of the above given matching engine inside Java based multi agent development frameworks is so easy. One such implementation has been tested on SEAGENT [7] and will be discussed later. However it should be noted that integration of the engine to the directory service(s) is an important issue and it needs an appropriate communication mechanism to realize semantic service discovery. Directory structures –e.g. directory facilitators in FIPA-compliant MASs- retrieve service discovery requests of the agents in MAS platform specific language structures –such as ACL (Agent

Communication Language) in FIPA MASs- and so, those requests should be pre-processed and transformed into the language of the semantic matching engine. In our design, the semantically enriched directory facilitator parses ACL content of an incoming service discovery request and prepares the appropriate semantic matching engine input (SeagentMatchRequest). After its execution, the engine outputs the matchmaking results in a predefined collection (SeagentMatchResults). Finally the directory facilitator puts those results into the outgoing ACL reply message and sends this message back to the requester agent. All those message conversions are performed inside the directory facilitator's communication module. During those message conversions, the module uses ontology mappings of the domain to generate OWL representations of the service requests and results to be used in message contents.

4 A Case Study on Semantic Capability Matching

To give ideas in a more concrete way, a case study on the proposed semantic capability matching is discussed in this section. We have tested the matching engine on a MAS that is developed by using SEAGENT [7] framework. The agent environment is about "Tourism" domain in which customer agents try to reserve hotel rooms on behalf of their users while some other agents are offering hotel services for those ones. For our simple scenario, four agents were created using SEAGENT framework and these agents registered their services to the semantically enriched directory facilitator (DF) of the system. For each service advertisement; the servicing agent's name and service type is given in Table 1.

Table 1. Four agent services registered into the directory facilitator

DFAgentDescription	Agent's Name	Service Type (from Tourism.owl)
D1	health@agents.com	HealthTourism
D2	summer@agents.com	SummerTourism
D3	sea@agents.com	SeaTourism
D4	tourism@agents.com	SummerTourism

Service type of each description is a concept defined in the domain ontology given in Fig. 2. Actually, service database of the DF is also an *ontology model* in which each agent service is represented with an individual. For example the DF description for the service offered by the agent called "health@agents.com" has a description on the model as follows (xml namespace definitions for RDF and OWL are omitted due to space limitations):

```
<rdf:RDF xmlns:j.0="http://.../~aegeants/ont/fipa-agent-management.owl#">
 <rdf:Description rdf:nodeID="D1">
   <rdf:type rdf:resource="fipa-agent-management.owl#DFAgentDescription"/>
   <j.0:name rdf:nodeID="A1"/>        <j.0:service rdf:nodeID="S1"/>
 </rdf:Description>
 <rdf:Description rdf:nodeID="A1">
   <rdf:type rdf:resource="fipa-agent-management.owl#AgentIdentifier"/>
   <j.0:name>health@aegeants.com</j.0:name>
   <j.0:addresses rdf:nodeID=" "/>        <j.0:resolvers rdf:nodeID=" "/>
  </rdf:Description>
  <rdf:Description rdf:nodeID="S1">
    <rdf:type rdf:resource=" fipa-agent-management.owl#ServiceDescription"/>
    <j.0:name>Health Tourism</j.0:name>
    <j.0:type>http://.../~aegeants/ont/Tourism.owl#HealthTourism</j.0:type>
   </rdf:Description>        </rdf:RDF>
```

”fipa-agent-management.owl” is an ontology that involves the concepts given in FIPA AMS [8]. It makes the DF descriptions fully compliant with FIPA specifications. Service types of the descriptions come from the “Tourism” domain ontology.

Then we developed a customer agent which has a simple plan looking for a hotel with “SummerTourism” service. However, in addition to exactly matched agent services, it requests other services semantically related with this type. Here is the OWL representation of the request parsed by the communication module and transmitted in a SeagentMatchRequest object to the engine (again namespace definitions are omitted):

```
<rdf:RDF xmlns:j.0="http://.../~aegeants/ont/seagent-match-ontology.owl#">
  <rdf:Description rdf:nodeID="R1">
    <rdf:type rdf:resource="seagent-match-ontology.owl#MatchRequest"/>
    <j.0:hasQuery rdf:nodeID="Q1"/>
    <j.0:hasSemanticMatch rdf:nodeID="S1"/>
    <j.0:hasPremise rdf:nodeID="P1"/>        <j.0:hasPremise rdf:nodeID="P2"/>
    <j.0:mustBindVariable>?x</j.0:mustBindVariable>
    <j.0:matchDegree>1</j.0:matchDegree>
  </rdf:Description>
  <rdf:Description rdf:nodeID="P1">
    <rdf:type rdf:resource="seagent-match-ontology.owl#Premise"/>
    <j.0:subject>?s</j.0:subject>
    <j.0:object>fipa-agent-management.owl#ServiceDescription</j.0:object>
    <j.0:predicate>http://www.w3.org/1999/02/22-rdf-syntax-ns#type</j.0:predicate>
  </rdf:Description>
  <rdf:Description rdf:nodeID="P2">
    <rdf:type rdf:resource="seagent-match-ontology.owl#Premise"/>
    <j.0:subject>?x</j.0:subject>
    <j.0:object>fipa-agent-management.owl#DFAgentDescription</j.0:object>
    <j.0:predicate>http://www.w3.org/1999/02/22-rdf-syntax-ns#type</j.0:predicate>
  </rdf:Description>
  <rdf:Description rdf:nodeID="Q1">
    <rdf:type rdf:resource="seagent-match-ontology.owl#Query"/>
    <j.0:subject>?x</j.0:subject>        <j.0:object>?s</j.0:object>
    <j.0:predicate>fipa-agent-management.owl#service</j.0:predicate>
  </rdf:Description>
  <rdf:Description rdf:nodeID="S1">
    <rdf:type rdf:resource="seagent-match-ontology.owl#SemanticMatch"/>
    <j.0:subject>?s</j.0:subject>
    <j.0:object>Tourism.owl#SummerTourism</j.0:object>
    <j.0:predicate>fipa-agent-management.owl#type</j.0:predicate>
  </rdf:Description>   </rdf:RDF>
```

Service discovery request is obviously an individual of the "seagent-match-ontology.owl" with the type of "MatchRequest". It both contains semantic matching and filtering query parameters which are given in RDF triple format. "Premise" properties of a request list RDF type of query variables while "Query" properties define relations between those variables. They may also include non-semantic query sentences those to be used in RDQL query. Every "SemanticMatch" property of the request defines a semantic query. "mustBindVariable" and "matchDegree" gives return type and desired match degree of the request respectively. Above "Premise" and "Query" descriptions tell that "?s" is a "ServiceDescription" and "?x" is a "DFDescription" and "?s" is a service of "?x". Finally, "SemanticMatch" property says that the request need a semantic query on *type* of the "?s": Find advertised agent services (ServiceDescriptions) whose types has a semantic relation with the type "SummerTourism". Match results will be of type "?x" (so DFDescription) and desired match degree is 1 (so SUBSUMES).

We examined that when the matching engine received this request; it first performed an RDQL query on the advertised agent services and filtered them according to the non-semantic parameters. Then, it used its reasoner (OWLOntolog) to determine semantic relationship between the given request and recently filtered service advertisements. The requester agent had asked for the services those have at least a "subsumes" relationship between the given request service type ("SummerTourism" in case). So, the engine matched the service descriptions D2, D3 and D4 with the given request as we expected and it sorted the match results starting from the most exact one(s) in the following order: D2, D4, D3 with EXACT, EXACT and SUBSUMES match degrees respectively. Each match result was returned to the communication module in a SeagentMatchResult object. According to our design, the DF used OWL representations of those results to put them into the ongoing ACL message and sent this result message to the requester agent. Then, the customer agent parsed the content and de-serialized each result object to proceed on its task. During this de-serialization it used "seagent-match-ontology.owl" to have an idea on ontology concepts those describes each match result due to match results are also semantically defined in the seagent-match-ontology likewise match requests. After all, the customer agent successfully retrieved *appropriate* services and it was ready to communicate agents offering those matched services.

5 Conclusion

Capability matching on agent services is a big challenge especially taking into consideration of future's semantic web enabled multi agent platforms. The main contribution of this study is to present a working model for semantic service matching on multi agent systems by indicating requirements to deal with that challenge. It introduces basics and design details of a semantic matching engine which is fully operational on MASs. Its integration into directory service structures is discussed with a working example given in this paper.

The engine is currently in use. However, we believe that it needs improvements on its capabilities taking into account of near future's platform requirements. Currently we are working on the Ontolog component of the engine and trying to enhance its reasoning system by adding support for extra ontology class relations such as "intersection" and "union". So, matching engine will be able to response more complex service discovery requirements (e.g. identifying composite agent services that owns two or more semantic types).

Our future work is to bring an abstraction to inner software design of the engine via a container mechanism such that it permits use of other semantic web tools in addition to (or in place of) Jena [11]. So, changes in these tools will not affect the core structure and it will need no or less modification on matching software.

References

1. Bellifemine, F., Poggi, A., and Rimassa, G.: Developing Multi-agent Systems with a FIPA-compliant Agent Framework, Software Practice and Experience, 31 (2001) 103-128
2. Berners-Lee, T., Hendler, J. and Lassila, O.: The Semantic Web, Scientific American, 284(5), (2001), pp:34-43
3. Chen, H., et al.: Intelligent Agents Meet Semantic Web in a Smart Meeting Room, in the proc. of Autonomous Agents and Multi Agent Systems 2004 (AAMAS'04), NY, USA
4. Connolly, D., et al.: DAML+OIL (March 2001) Reference Description, available at: http://www.w3.org/TR/daml+oil-reference
5. Cost, R. S., et al.: Jackal:A Java-Based Tool for Agent Development, in the proc. workshop tools for Developing Agents (AAAI98), AAAI Press, Calif., (1998), pp:73-82
6. Cost, R. S., et al.: Ittalks: A Case Study in the Semantic Web and DAML+OIL, IEEE Intelligent Systems, January-February, (2002), pp:40-46
7. Dikenelli, O., et al.: SEAGENT: A Platform for Developing Semantic Web Based Multi Agent Systems, accepted to be presented in Fourth International Joint Conference on Autonomous Agents and Multi Agent Systems 2005 (AAMAS'05), Utrecht, The Netherlands
8. FIPA (Foundation for Intelligent Physical Agents): FIPA Specifications, available at: http://www.fipa.org
9. Gomez, M., and Plaza, E.: Extending matchmaking to maximize capability reuse, in the proc. of Autonomous Agents and Multi Agent Systems 2004 (AAMAS'04), NY, USA
10. Graham, J. R., Decker, K. S. and Mersic, M.: DECAF – A Flexible Multi Agent Systems Infrastructure, Journal of Autonomous Agents and Multi-Agent Systems,7 (2003), 7-27
11. JENA - A Semantic Web Framework for Java, available at: http://jena.sourceforge.net
12. Li, L. and Horrocks, I.: A Software Framework for Matchmaking based on Semantic Web Technology, in the proc. of WWW'2003, Budapest, Hungary, pp. 331-339
13. McGuiness, D. L., and van Harmelen, F.: OWL Web Ontology Language Overview, (2004), available at: http://www.w3.org/TR/owl-features/
14. Sycara, K., Paolucci, M., Ankolekar, A., and Srinivasan, N.: Automated discovery, interaction and composition of Semantic Web Services, Journal of Web Semantics, Elsevier 1, (2003) pp. 27-46
15. The DAML Services Coalition: DAML-S 0.9 (May 2003): Semantic Markup for Web Services, available at: http://www.daml.org/services/daml-s/0.9/daml-s.html
16. Zambonelli, F., Omicini, A.: Challenges and Research Directions in Agent-Oriented Software Engineering, Journal of Autonomous Agents and Multi-Agent Systems, Vol. 9, No. 3, (2004)