An agent Based Approach to Web service Discovery

Berdjouh Chafik
Centre de Formation Professionnelle El-Meghaier
Wilaya El-OUED, ALGERIE
Berdjouh2006@yahoo.fr,

Kazar Okba
Département d'informatique
Faculté des sciences et sciences de l'ingénieur
Université Mohamed Khider 07000 Biskra, ALGERIE
kazarokba@yahoo.fr

ABSTRACT

Web services are emerging and promising technologies for the development, deployment and integration of Internet applications. They are based on three main bricks that are SOAP (Simple Object Access Protocol), WSDL (Web Service Description Language) and UDDI (Universal Description, Discovery and Integration). The language used behind these protocols is XML (eXtensible Markup Language), which makes Web services independent of platforms and programming languages. They have become very effective in the interoperability of systems. The need to introduce semantics in Web services is felt to automate the different phases of their life cycle, namely the discovery phase.

The concept of semantic web services, is the result of convergence in the field of web services with the Semantic Web, in fact its ultimate goal is to make web services more accessible to the machine by automating tasks that facilitate their use. In this work, we study the problem of semantic discovery of services by providing a method that is based on agents.

Key Words: Web service, multi-agents system, semantic web

1. Introduction

Today, the Web is just an enormous warehouse of text and images, its development has also made it a service provider. The concept of "Web service" is essentially an application available on the Internet by a service provider and accessible by clients through standard Internet protocols. In essence, Web services are autonomous software components and self-descriptive and thereby constitute a new paradigm for application integration.

Currently, Web services are implemented through three technology standards: WSDL, UDDI and SOAP. These technologies facilitate the description, discovery and communication between services. However, this basic infrastructure does not yet allow Web services to keep their promise of a largely automated management. This automation is essential to meet the requirements to scale and to reduce development costs and maintenance services. Basically, it must accommodate a means for describing Web services in a manner understandable by a machine.

The Semantic Web [1] is a vision of future Web in which information has a semantic understandable by a computers. Applied to Web services, the principles of the Semantic Web should enable to describe the semantics of their functionality, and reasoning are therefore induced a proposal for automation of various tasks of their life cycle. Combining the technologies of Web services and Semantic Web has led to the concept of semantic Web services.

The discovery of Web services is an emerging area of research. Initially, the discovery is made in the UDDI registry, it is based primarily on research syntactic WSDL descriptions of Web services. But with the development of Semantic Web technologies, the techniques for discovery have become essentially semantic. This semantics is provided through one of ontologies important technologies of the Semantic Web. Thus, software agents can be developed to reason about these ontologies, making the discovery of Web services dynamic and automatic.

In this work, we propose an approach to discovery of semantic web services using agent technology and ontologies.

2. Emerging Technologies

2.1 Semantic Web and ontology

The Semantic Web [1] is an extension of the current Web in which information is given well-defined meaning, better enabling computers and people to work in cooperation. In order to realize the Semantic Web vision a set of standard technologies have been defined (the Semantic Web layered architecture):

- the Syntactic Layer (XML),

- the metadata layer (RDF/RDFS),
- the semantic layer (Ontology languages),
- the Logical Layer (automatic reasoning),
- Proof and trust layer (proof)

The technology object which is fundamental for the realization of the Semantic Web is the ontology.

Ontology is term borrowed from philosophy meaning "systematic explanation of existence". Ontology is similar to a dictionary or glossary but with a large and detailed structure, that allows machines to process its contents.
Bertrand [2] defines ontology as ”These are formal representations of domain knowledge in the form of terms with semantic relations”.

In the Semantic Web, the ontology allows the user during a Web search to access not only to documents related to keywords in the query, but also those that are related ontologically (semantically) to them, this makes the search more relevant. It aims to describe concepts and relationships that bind them, and with deduction rules to make them more understandable and usable by the different agents (human or software).

2.2 Ontology Web Language for Services (OWL-S)

OWL-S (formerly known as DAML-S) [3] is ontology for web services and it has been developed to enable the following tasks: automatic service discovery, automatic service invocation and automatic service composition. The service discovery is improved using ontologies because the information needed to perform this task is expressed using a machine-processable form. A computer can access the description of a web service and it can know exactly what the service does thanks to the shared concepts contained in the ontologies used in the description.

A service described using OWL-S provides three types of knowledge: Service Profile, Process Model and Service Grounding. The ServiceProfile describes what the service does, including functional information such as inputs, outputs, and other non-functional information (category, classification). It is normally used during the automatic discovery of Web services. The Process Model describes how the service works; it is an abstract vision of the service operation. Finally, the ServiceGrounding tells how to access the service; it contains all the information related to the real implementation of the service and is used to invoke it automatically.

2.3 Role of Multi-Agent Systems

A multi-agent system (MAS) is a loosely coupled network of software agents that interact to solve problems and function beyond the capabilities of any singular agent in the set-up. The agents in a multi-agent system may be distributed on different computers (or nodes), where each computer owns its resources.

The characteristics of MAS are that each agent has incomplete information or capabilities for solving the problem and, thus, has a limited viewpoint; there is no system global control; data are decentralized; and computation is asynchronous[4].

An MAS has the following advantages over a single agent or centralized approach like distributed systems [5]:

• An MAS distributes computational resources and capabilities across a network of interconnected agents. Whereas a centralized system may be plagued by resource limitations, performance bottlenecks, or critical failures, an MAS is decentralized and thus does not suffer from the "single point of failure" problem associated with centralized systems.

• An MAS allows for the interconnection and interoperation of multiple existing legacy systems. By building an agent wrapper around such systems, they can be incorporated into an agent society.

• An MAS models problems in terms of autonomous interacting component-agents, which is proving to be a more natural way of representing task allocation, team planning, user preferences, open environments, and so on.

• An MAS efficiently retrieves, filters, and globally coordinates information from sources that are spatially distributed.

• An MAS provides solutions in situations where expertise is spatially and temporally distributed.

• An MAS enhances overall system performance, specifically along the dimensions of computational efficiency, reliability, extensibility, robustness, maintainability, responsiveness, flexibility, and reuse.

3. Our Approach

The proposed architecture is an extension of service-oriented architecture (SOA). This architecture is based on agents for discovering Web services.

This architecture (shown in Figure 1) incorporates software components and operating a domain ontology is used during the discovery phase of Web services, it facilitates the automatic discovery of services since it allows to refine the search process which matches a request and service offerings. The use of this ontology allows the implementation of filtering mechanisms (comparison) between a request and offers to implement anything other than simple equality.

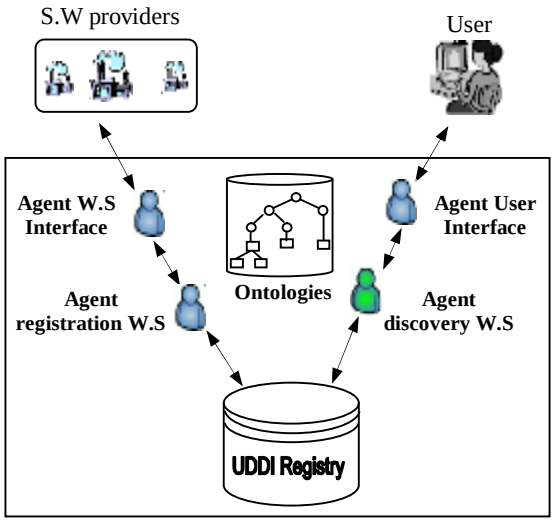

Fig. 1. Proposed multi-agent architecture

3.1 Architecture description

3.1.1 Agent Web service interface

This software agent acts as an interface between the system and the Web service provider, such that for each Web service agent is associated. Agent Web service interface allows the recording of the description on the Semantic Web service. Moreover, it allows updates information on the Web service.

The internal architecture of the agent Web service interface consists of three modules and a registry backup, as shown in Figure 2.

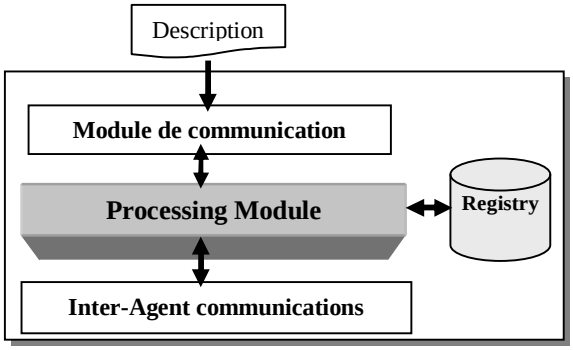

Fig. 2. Architecture of the Web service interface agent

3.1.2 Agent registration Web services

The role of this agent is the preservation of semantic descriptions of Web services in the UDDI registry, it contains two modules and an interface as shown in Figure 3.

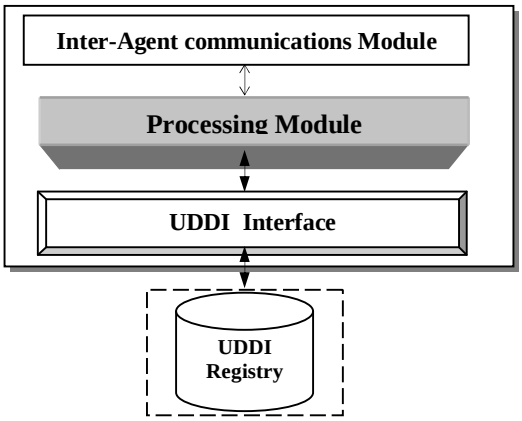

Fig. 3. Architecture of agent registration Web services

3.1.3 Agent user interface

The agent user interface is the gateway to query the system. It provides the user with the form to do a query.

This is the agent who will initiate the discovery, by issuing to the Agent discovery, a request consists of inputs, outputs, a reference ontology domain to use (e.g. the ontology of tourism) and presents the results tailored to the preferences of users after treatment.

The internal architecture of the agent user interface is composed of three main modules and a registry backup as shown in Figure 4.

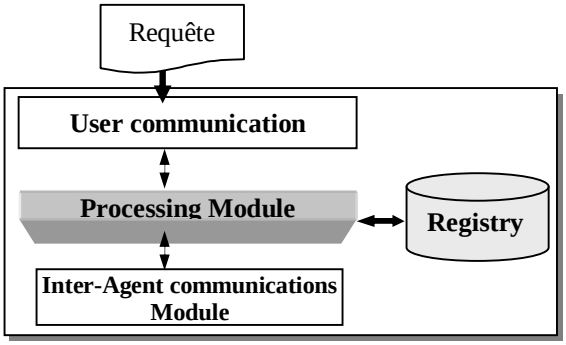

Fig. 4. Architecture de agent user interface

3.1.4 Agent discovery Web services

It is a software agent that allows the discovery of descriptions of Web services satisfying the request sent by the agent user interface on the semantic.

The internal architecture of the discovery agent is composed of two modules and a base of storage services for storage the semantic descriptions of services provided by UDDI as shown in Figure 5. They are as follows:

- **Inter-Agent communications Module :** He received from the agent user interface the query in the form of a message and after that, he calls the module of treatment. It also receives requests for transmission of messages from the module of treatment. Such requests for transfers are received answers queries.
- **Base of storage Services** : is used to store the semantic descriptions of Web services satisfying the user query.

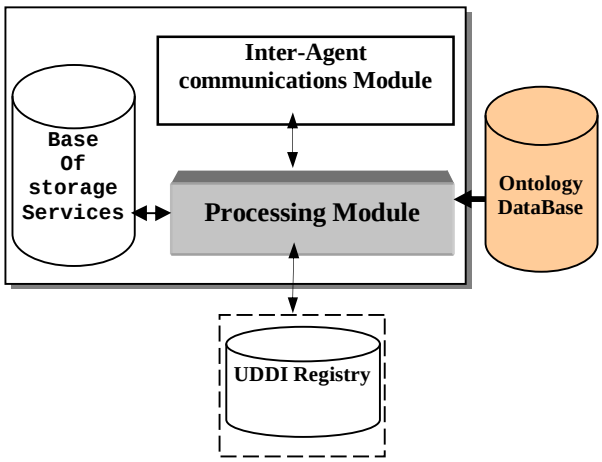

Fig. 5. Architecture of agent Discovery Web service

- **Processing Module**: it has two tasks:

1) *The task of analysis*: selects the domain ontology corresponding to the request (from base ontology that stores ontologies in various domains), extract the classes and their links and builds the corresponding tree. In our context, this action is possible since the vocabulary defined in the domain ontology is described in a hierarchical form. Each vertex of this tree corresponds to a class of the ontology and each arc corresponds to a subclass. This tree used to infer relationships of generalization (subsumption) between the concepts, i.e. that a concept is more general than another. A concept C includes (subsumes) a concept C ' if the extension of C' is included in that of C. Then we say that C is more general than (or includes) C '. This principle allows us to make comparisons between flexible offers and requests.

2) *The task of comparison*: You can compare applications and service offerings by considering the ontology (see Figure 6) in accordance with four main modes of comparison defined in [6] using a matchmaking algorithm: Exact mode, Plug-In mode, subsumes mode and Fail mode.

1. Exact mode selects an offer if it corresponds exactly to a request (demand = offer) i.e. inputs and outputs of the offer is equivalent to the input and output of demand (matching exact);

2. Plug-in mode returns an offer if it includes a request (demand <offer) ie the entries in the application includes the supply of inputs and outputs of the application are covered by the output of supply in the domain ontology (inclusive matching);

3. Subsumes mode returns an offer if it is included in a request (demand > offer) (the inverse of plug-in mode) (partial matching);

4. Fail method returns false if no match between offer and demand (demand # offer) (matching failure).

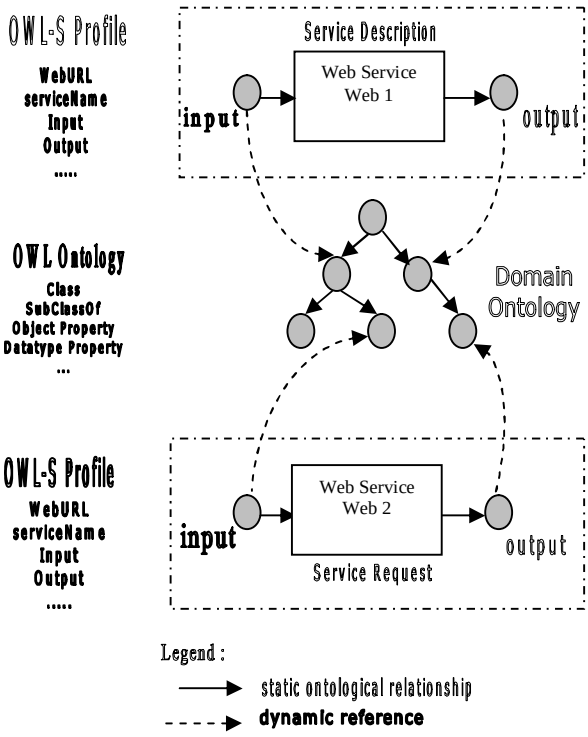

Fig. 6. methodology for generating semantic descriptions of Web services

Modes 2 and 3 for comparison using the domain ontology. More specifically, the offers and requests (demands) for services are expressed in OWL-S, we compare, according to the four modes, all the elements defined in the terms "input" and "output" in the class ServiceProfile of offers and requests (demands).

The comparison algorithm used in both plug-in mode and subsumes mode uses the function Includes [7] (see Figure 7).

The agent uses a subsumption test on outputs (outputs) (see Figure 8) then we assign a score for each mode matching: Exact (score = 3), Plug-In (score = 2) subsumes (score = 1), Fail (score = 0) (see Figure 9).

```
Function Includes (E1 : string, E2 : String) : boolean
% This function returns true if E1 includes E2 false else
% E1 is an element of the clause Input or Output of Offer
% E2 is an element of the clause Input or Output of demand
% A represent the ontology (tree form)
% We uses the high level functions include :
% Father(E) : returns the father of E in A

% Root(A) : returns the root of A
Variables
SummitOngoing : ASummit   % A summit under discussion
TheAncestors: SetofSummits % The Ancestors of E2
Begin
TheAncestors ß ∅
If E2 = root(A) Then
% E2 does not an ancestor and can not be subsumed
TheAncestors ß ∅
Else
SommetCourant ß Father(E2)
TheAncestors ß Father (E2)
While (SommetOngoing <>Root(A)) Do
        SummitOngoing ß Father(SummitOngoing)
         % «+» means adding a new element
         % throughout the Ancestors
        TheAncestors ß TheAncestors +
  SummitOngoing
End While
End If
        Includes ß (E1 ∈ TheAncestors)
End
```

Fig. 7. Function Includes

```
Function GetScore(rel : String) : Integer
% This Function returns the score of matching
Val =0
Begin
    If rel = "Exact"   Then  val = 3
    If rel = "PlugIn"  Then  val = 2
    If rel = "Subsume" Then  val = 1
    If rel = "Fail"    Then  val = 0
    End If
    Return val
End
```

Fig. 8. Procedure of matching for the outputs

```
Procedure degreeOfMatch(OutD,OutO : String )
% This Procedure returns result of comparison
% OutD, OutO are the output of demand and offer respectively
Begin
  If  OutO = OutD Then Return Exact
  If  Includes(OutO, OutD) Then  Return PlugIn
  If  Includes(OutD, OutO) Then  Return Subsume
  Otherwise Return Fail
  End If
End
```

Fig. 9. Function returns the score of matching

The matching between inputs is computed following the same procedure.
Equation (1) generalize the comparison between a service concept C_i^O and a corresponding request concept C_i^D :

$$match(C_i^D, C_i^O) = \begin{cases} 3 & If\ C_i^D = C_i^O \\ 2 & If\ C_i^D \sqsubset C_i^O \\ 1 & If\ C_i^D \sqsupset C_i^O \\ 0 & Else \end{cases} \quad (1)$$

Assuming there are m concepts in a service description and there are m corresponding concepts in a service request, the similarity or global match between the request R and the service S can be derived by summing up the match scores between the a concept pair (equation (2)) :

$$Similarity(D, O) = \sum_{i=1}^{m} Match(C_i^D, C_i^O) \quad (2)$$

Therefore, the matching between a request and a set of services can be quantitatively measured. A service with the highest similarity score represents the most accurate service for the request. There may be more than one most accurate service. Besides the most accurate service(s), those services with a similarity greater than zero are still useful as backup services.

3.2 Illustrating example

Assume that there are three Web services sales : S1, S2 and S3 published on the Web. Functional parameters (inputs, outputs) are:

- S1 have two inputs "vehicle" and "parts", and one output "price".
- S2 have two inputs "parts" and "car" ,and one output "price".
- S3 have two inputs "unit" and "material" and one output "price".

Consider a user request R contains two inputs "Car" and "Parts" and one output "Price"
Given the ontology fragment shown in figure 10.

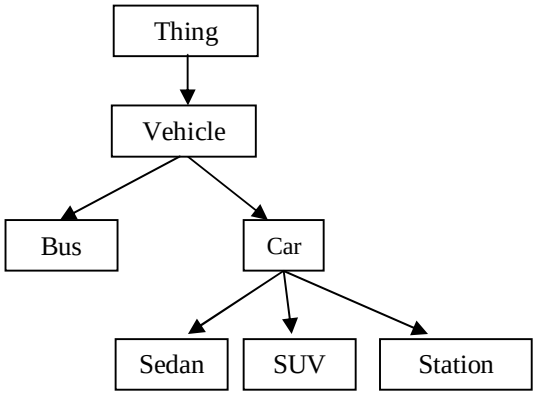

Fig. 10. A fragment of the Vehicle ontology

If we apply the matching algorithm, we obtain the following results:

- Comparison of inputs :

S1:
Car **à** vehicle, Founded Input Relation is = Plug-in, Their score = 2, Total Score is : 2
Car **à** parts, Founded Input Relation is = Fail, Their score = 0, Total Score is: 2
Parts **à** vehicle, Founded Input Relation is = Fail, Their score = 0, Total Score is: 2
Parts **à** parts, Founded Input Relation is = Exact, Their score = 3, Total Score is: 5
Inputs score weight = 5

S2:
Car **à** parts, Founded Input Relation is = Fail, Their score = 0, Total Score is: 0
Car **à** car, Founded Input Relation is = Exact, Their score = 3, Total Score is: 3
Parts **à** parts, Founded Input Relation is = Exact, Their score = 3, Total Score is : 6
Parts **à** car, Founded Input Relation is = Fail, Their score = 0, Total Score is : 6
Inputs score weight = 6

S3 :
Car **à** unit, Founded Input Relation is = Fail, Their score = 0, Total Score is : 0
Car **à** material, Founded Input Relation is = Fail, Their score = 0, Total Score is : 0
Parts **à** unit, Founded Input Relation is = Fail, Their score = 0, Total Score is : 0
Parts **à** material, Founded Input Relation is = Fail, Their score = 0, Total Score is: 0
Inputs score weight = 0

- Comparison of outputs :

S1 : price **à** price, Founded Output Relation is = Exact, Their score = 3, Total Score is : 3
Outputs score weight = 3
S2 : price **à** price, Founded Output Relation is = Exact, Their score = 3, Total Score is : 3
Outputs score weight = 3
S3 : price **à** price, Founded Output Relation is = Exact, Their score = 3, Total Score is : 3
Outputs score weight = 3

- global matching:

S1: Total score (total score for inputs + total score for outputs) = 5 + 3 = 8, Good
S2: Total score = 6 + 3 = 9, Best
S3: Total score = 0 + 3 = 3, Not Good

► Thus, the Web service S2 is regarded as the best corresponding to the request.

4. Conclusions and Future Work

In this paper we presented a conceptual framework and architecture based on Web services for interoperability.

The discovery of Web services is an emerging area of research. Various approaches have been proposed. These approaches have shifted from a search based keywords (discovery syntactic) to methods based semantics. We proposed an approach based on agents that models the discovery of semantic web services. Our architecture based agents consists of :

-An agent Web services interface;
-An agent user interface;
-An agent registration Web service;
-An agent Web service discovery.

Agent Web services discovery apply inferences to match the user query with the services offered. The pairing (matching) based on comparing the outputs and inputs of the request with the outputs and inputs of the service, and presents different levels of matching: exact, plug-in, subsume and fail.

In the short term, we will implement our proposed architecture. To validate our work, we will conduct tests with a variety of user queries and a panel of Web services.

Regarding the prospects for our work, we expect the following:

- As regards the matching algorithm could provide for other search parameters such as preconditions and effects, they increase the rate of accuracy.
- Submit an indirect matching in the absence of direct matching i.e. move to Web services composition.
- We can try to use other types of agents such as mobile agents and assess their effects on performance.

References :

[1] Berners-Lee, T., Hendler, J. et Lassila, O., The Semantic Web, In Scientific American, vol. *284* No. 5, May 2001, pp. 35 - 43.

[2] Bertrand Sajus, "La fonction Thésaurale au coeur des systèmes d'information" ADBS, Avril 2002, **www.adbs.fr/adbs/prodserv/jetude/html/prog110402a.html**.

[3] W3C, OWL-S Semantic Markup for Web Services, W3C Member Submission, Novembre 2004, **http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/**.

[4] Katia P. Sycara, Multiagent Systems,AI magazine, Volume 19, 1998.

[5] Multi-Agent Systems, The Intelligent Software Agents Lab, 2008, **http://www.cs.cmu.edu/~softagents/multi.html**.

[6] Paolucci, M., Kawamura, T., Payne, T., Sycara, K.,Semantic matching of web services capabilities. In: Proceedings of the First International Semantic Web Conference, LNCS 2342, Springer-Verlag, 2002, pp. 333 – 347.

[7] Bouzguenda L., «Coordination Multi-Agents pour le Workflow Inter Organisationnel Lâche», Thèse de Doctorat, Université de Toulouse 1, mai 2006, pp. 100 - 102.