P2P support for OWL-S discovery

Domenico Redavid[†], Stefano Ferilli[⋆], and Floriana Esposito[⋆]

[†]Artificial Brain S.r.l., Bari, Italy
[⋆]Computer Science Department, University of Bari "Aldo Moro", Italy
redavid@abrain.it
{ferilli, esposito}@di.uniba.it

Abstract. The discovery of Web services is often influenced by the rigid structure of registries containing their XML description. In recent years some methods that replace the traditional UDDI registry with Peer To Peer networks for the creation of catalogs of Web services have been proposed in order to make this structure flexible and usable. This paper proposes a different view by placing the semantic description of services as content of P2P networks and showing that all the needed information for an efficient Web service discovery is already contained in its OWL-S description.

1 Introduction

The discovery of Web services (Ws)[1] is achieved through Universal Description, Discovery and Integration (UDDI)[2], which provides a standard mechanism to register and search WS descriptions. An UDDI registry is an indexed database that contains instances of Web Services Description Language (WSDL)[3] in turn based on eXtensible Markup Language (XML)[4] and independent from hardware platforms. A requester needing to use a service queries the UDDI registry to find the one that best meets its needs. The register returns an access point and WSDL description which are then used by the requester to build needed SOAP[5] messages to communicate with the actual service.

The UDDI registry is supported by a worldwide network of nodes connected in a federation. When a client sends information to a registry, this is propagated to other nodes. In this way it implements data redundancy, providing a certain degree of reliability. However, data replication implies lower consistency and is not a scalable approach. Another limitation of UDDI is the search mechanism: it can focus only on a single search criterion such as name, location, category

[1] W3C Web of Services - `http://www.w3.org/standards/webofservices/`
[2] Universal Description, Discovery and Integration v3.0.2 (UDDI), OASIS Specification - `http://uddi.org/pubs/uddi_v3.htm`
[3] Web Services Description Language (WSDL) Version 2.0 Part 0: Primer, W3C Recommendation 26 June 2007 - `http://www.w3.org/TR/wsdl20-primer/`
[4] W3C XML Technology - `http://www.w3.org/standards/xml/`
[5] SOAP Version 1.2 Part 0: Primer (Second Edition), W3C Recommendation 27 April 2007 - `http://www.w3.org/TR/soap12-part0/`

of business, etc.. Within the Service-Oriented Architecture (SOA) [3], the register has a role similar to yellow pages where a list of services can be found. To fully exploit the potential of this type of architecture, the register should be consultable not only by humans but also by software systems that need to find, select and compose services in an automatic way. In recent years research has been focusing on Peer-to-Peer (P2P) technologies [2] that offer Distributed Hash Table (DHT) [11] functionalities. A P2P network provides a typical distributed decentralized approach where multiple computers are interconnected and communicate by exchanging messages. DHT partitions the items of a key set among participating nodes, and can send messages to the owner of a given key in an efficient manner. The P2P network with DHT support is scalable and solves the problem of data redundancy, but supports only exact match for keywords. The inclusion in the P2P network of references to service semantics could be a turning point, because such information could be exploited for the automatic discovery of services with the help of semantic matchmaking techniques. In this paper we discuss how this vision can be realized. Section 2 introduces the basic concepts related to WS registries and Catalogs based on P2P protocols, and the OWL-S language for the representation of the semantics associated with services. Section 3 describes an implementation of the P2P network created by means of the Open Chord API and OWL-S. Finally, Section 4 presents an analysis on the potential of the proposed approach.

2 Background

2.1 Web service registries

Web services are software systems identified by means of a Web address and designed to support interoperability between computers on a network. They have public interfaces defined and described as XML documents in a format, such as WSDL, that can be processed by a machine in an automatic way. Their definitions can be sought from other software systems, which can directly interact with the Web service operations described in the interface by activating the appropriate messages enclosed in an SOAP envelope. These messages are usually transported via the Hyper Text Transport Protocol (HTTP) and formatted according to the XML standard. For the purposes of this discussion is important to point out what are the current approaches to the organization of Web services. These approaches can be broadly classified as centralized or decentralized. The traditional centralized approach includes UDDI, where a central registry is used to store descriptions of Web services. The current UDDI approach attempts to mitigate the disadvantages of centralization by replicating the entire information on different sites. Replication, however, may temporarily improve performance if the number of UDDI users is limited. But with the rise of the replicated sites, decreases the consistency of the duplicated data. The replication of UDDI data is not a scalable approach. For this reason, different approaches on decentralized registries have been proposed in order to connect individual customers through the P2P network. Since this technology organizes peers into a hypercube, the management becomes inefficient in the presence of a large amount of data. A solution to this problem is given by [14] where a method for the reduction in size of the indexing scheme that maps the multidimensional information space with physical peers is presented. However, this method does not use semantic descriptions. The Web Service discovery based on P2P is also discussed in [18] and [5] where ad hoc model frameworks are proposed for this purpose. As a starting point, in the next section we will analyze a proposed approach that combines ontologies and P2P based on DHT as a sophisticated solution to these issues.

2.2 Web Service catalog system based on DHT

Without a central registry, the easiest way to find out the location of a service in a distributed system is to send the query to each participant (service provider). While this approach might work for a small number of service providers, it is certainly not scalable in a large distributed system. When a system includes thousands of nodes, facilities that allow the selection of a subset of nodes that will be fitted with the functionality exposed in the catalogs are needed. The new generation of P2P systems includes complete DHT features [11] for decentralized applications. Some groups have proposed innovative approaches such as CAN [11], Pastry [13] and Chord [17], which eliminate the defects of the first P2P systems like Gnutella[6] and Napster[7]. Although they are implemented in different ways, all these systems have interfaces to support access to DHT. These interfaces permit to request shared information. In contrast to UDDI, the P2P network content is usually described and indexed locally within each peer, while search queries are propagated through the network. A central index that spans the whole network is not required. Given a key, the corresponding data items can be efficiently located using up to $O(log(n))$ network messages, where n is the total number of nodes in the system [17]. In addition, distributed systems evolve while remaining scalable to a large number of nodes. Current efforts are directed towards this functionality in order to provide a catalog of services fully distributed and scalable. The approach chosen for the purposes of this paper is Chord because it proposes an original approach to the problem of efficient location and is able to maintain the bandwidth close to the optimal in the management of arrival and departure of competing nodes [8].

Chord uses routed queries to locate a key, minimizing the number of visits on large amount of nodes. What distinguishes Chord from other P2P applications is the ease of use and provable performance and correctness. In essence, Chord supports one operation: given a key, it is mapped on a node. The location data can be implemented by associating each key with a datum. In detail, it routes a key through a sequence of $O(log(n))$ other nodes toward the destination. This requires that a Chord node has information about $O(log(n))$ other nodes for efficient routing. When the information is out of date, it is proved that the performance degrades gracefully. This is important in practice because nodes will

[6] Gnutella Web site - `http://rfc-gnutella.sourceforge.net/`

[7] Napster Web site- `http://free.napster.com/`

join and leave arbitrarily, and consistency about $O(log(n))$ nodes may be hard to maintain. Only one piece of information per node needs to be correct in order to guarantee correct routing of queries. The Chord protocol uses the $SHA-1$ hash functions to assign a *m-bit* identifier to each node and key. Furthermore, it uses Consistent Hash functions that allow nodes to leave and enter the network with minimal disruption [6]. The integer m is chosen to be large enough to make negligible the probability that two nodes (keys) received the same identifier. The hash function calculates the key identifier performing hashing over the IP address of the node. The key identifiers of the nodes are arranged in a circular ring of dimension $2m$ called Chord ring. The identifiers on the Chord ring are numbered from 0 to $2m-1$. A key is assigned to a node whose identifier is equal to or greater than the key identifier. This node is called the successor of the node k and is the first node k on the circle in a clockwise direction. When a node n wants to find a certain key k, it uses a lookup function that returns the successor of n if k is between n and its successor, otherwise forwards the query in the circle. Furthermore, in order to provide more efficient lookup, parts of the routing information are stored in the nodes. In particular, each node n maintains a routing table with at most m entries (where m is the number of bits in the identifiers), called finger table. Stabilization primitives are used to maintain updated finger tables, as well as the Chord ring itself.

This structure can be used to create Web Service Catalogs. For example, in [19] each node in the system is a service provider or requester so that both these two actors are connected together in the Chord ring. When a service provider N_i wants to publish a service, creates the service catalog item, i.e., the tuple $C = key, Summary$. The Chord protocol routes the catalog information to the corresponding node of the system in accordance with the key in the catalog. Thus, each node in the system contains part of information of catalog and all the nodes together constitute the global catalog system implementing the functionality of a traditional UDDI registry. With WSDL, a Web Service can be expressed as a set of operations, each of which implements a certain amount of functions. An operation is specified by its name and the types of input and output messages. The service name is used as the key catalog information for the DHT hashing algorithm. In line with this, the operations included in the service and messages associated with these operations are used as a summary. The catalog for a Web service Ws_i has the structure: $CWs_i = (Key, Summary, N)$, where:

- Key is the name of Ws_i,
- $Summary$ contains the operations and its messages included in Ws_i,
- N is the node that publishes Ws_i.

In the same paper [19] is proposed a mapping between the information contained in the nodes (related to the WSDL) with ontology classes that represent them in OWL-S services. In contrast, our approach foresees that the information contained in the catalogs are taken directly from the OWL-S descriptions already available online.

2.3 Web Ontology Language for Services (OWL-S)

Semantic Web Services[9] provide an ontological framework for describing services, messages, and concepts in a machine-readable format, enabling logical reasoning on service descriptions. The Web Ontology Language for Services (OWL-S) provides a Semantic Web Services framework on which an abstract description of a service can be formalised. It is an upper ontology described with OWL[8] whose root class is *Service*, therefore, every described service maps onto an instance of this concept. The upper level Service class is associated with three other classes:

Service Profile. The service profile specifies the functionality of a service. This concept is the top-level starting point for the customizations of the OWL-S model that supports the retrieval of suitable services based on their semantic description. It describes the service by providing several types of information:

- Human readable information: such as the service description, service name, contact information, etc.;
- Functionalities: i.e. parameter type identifiers, identifiers for the input and output, parameters of service methods, preconditions, results, etc.;
- Service parameters: which include parameter identifiers (e.g. name, value) used by the service;
- Service categories: these include identifiers for defining the category of service, i.e. category name, taxonomy, value, code;

Service Model. The service model exposes to clients how to use the service, by detailing the semantic content of requests, the conditions under which particular outcomes will occur, and, where necessary, the step by step processes leading to those outcomes. In other words, it describes how to ask for (invoke) the service and what happens when the service is carried out. From the point of view of the processes, the service model defines the concept *Process* that describes the composition of one or more services in terms of their constituent processes. A *Process* can be atomic, composite or simple: an atomic process is a description of a non-decomposable service that expects one message and returns one message in response. A composite process consists of a set of processes within some control structures defining a workflow. Whereas a simple process provides a service abstraction that allows to view a composite service as an atomic one. Each process can have any number of inputs, a set of preconditions, all of which must hold for the process to be successfully invoked, and any number of results (outputs and/or effects) that come from a successful execution of the service.

Service Grounding. A grounding is a mapping from an abstract to a concrete specification of those service description elements that are required for interacting with the service. In general, a grounding indicates a communication protocol, a message format and other service-specific details (e.g., port

[8] OWL Web Ontology Language, W3C Recommendation 10 February 2004 - `http://www.w3.org/TR/owl-features/`

numbers, the serialization techniques of inputs and outputs, etc.). From the point of view of processes, a service grounding enables the transformation from inputs and outputs of an atomic process into a concrete atomic process grounding constructs.

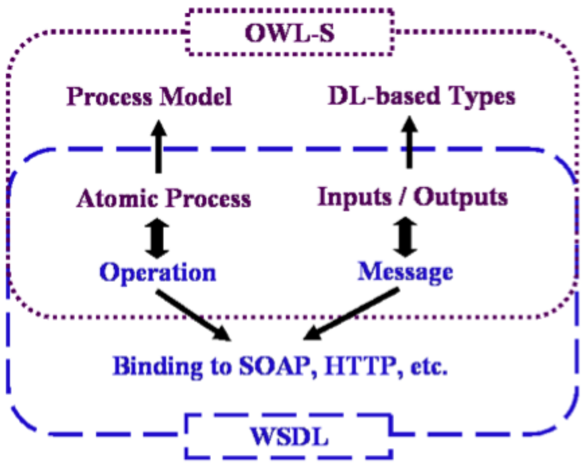


Fig. 1. Schema Mapping WSDL-OWL-S

As we can see from Figure 1, OWL-S grounding maps the semantic description of the service with the corresponding WSDL. This means that the information contained in the *Summary* of Catalog shown in the previous section can be directly obtained from OWL-S. Since each OWL-S instance, as well as all its constituent parts, has its own URI, such information is always available online.

3 OWL-S discovery with P2P

3.1 Open Chord

Open Chord[9] is an open source implementation of Chord. Its architecture consists of three levels (Figure 2). On the lower level is located the implementation of the communication protocol used (Communication Layer), based on a network protocol (such as Java Socket). Currently, two implementations are provided: *Local communication protocol*, that has been developed for testing purposes, and *Socket-based protocol*, that facilitates reliable communication between Open Chord peers based on TCP/IP sockets.

[9] Open Chord Web site - `http://open-chord.sourceforge.net/`

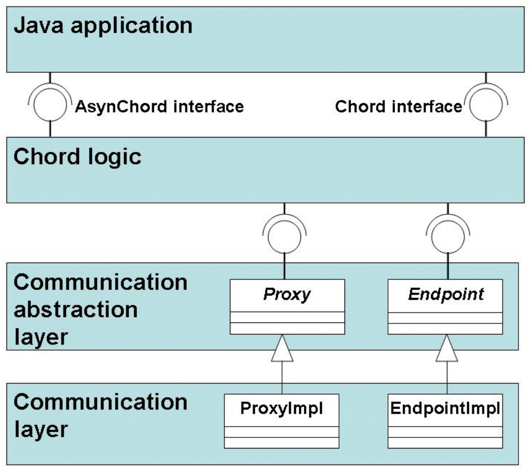


Fig. 2. Open CHORD architecture

The abstraction level (*Communication abstraction layer*) provides two abstract classes that must be implemented to realize the communication protocol:

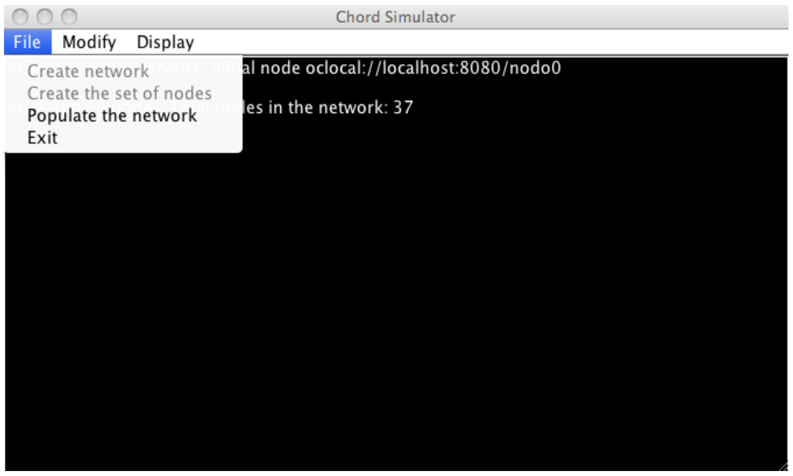
- *Proxy*, that represents a reference to remote peers in the Open Chord overlay network.
- *Endpoint*, that provides a connection point for remote peers conform to a specific communication protocol.

Concrete implementations for a communication protocol are determined with help of the URL of a peer.

The *Chord logic* level, which implements the functionality of Chord, offers two interfaces for Java applications that abstract from the implementation of the Chord DHT routing. Both interfaces (i.e., *Chord* and *AsynChord*), which can be used by an application built on-top of Open Chord to retrieve, remove, and store data in synchronous and asynchronous way from/to the underlying DHT, provide some common methods that are important to create, join, and leave an Open Chord DHT. The *Chord logic* level is also responsible for data replication and maintenance of the necessary properties to keep running the DHT, as described in [17].

3.2 A prototype implementation

To simulate an OWL-S P2P network using the Open Chord API a simple graphical application that displays a drop down menu consisting of the items *File*, *Edit*, and *View* has been developed. By selecting 'Create Network' from 'File' menu a



al node oclocal://localhost:8080/nodo0

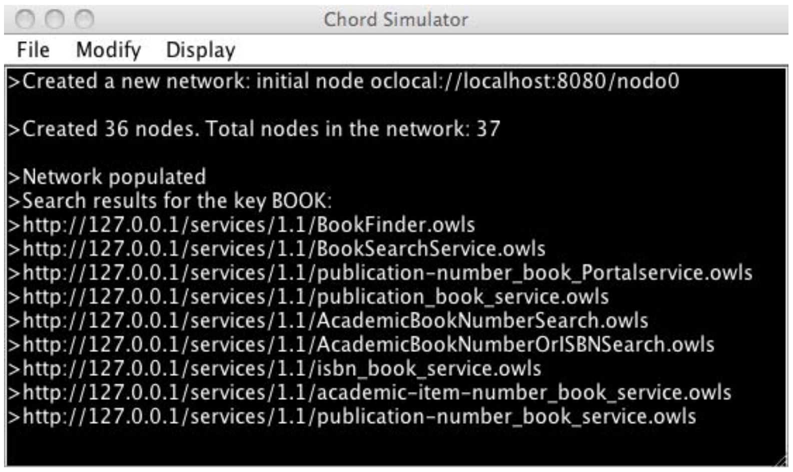
es in the network: 37

Fig. 3. Screenshot of the prototype

single peer will be created, consisting simply of a new URL. Only the first node has the ability to create a new network, to add other nodes the join method of the interface Chord will be invoked. This method works similarly to the method used to create the network, but in addition to the node that is to be added, an existing URL, that is already part of the network, is required. This is called bootstrap peer. To test the operations of the P2P network we have taken a set of services from the dataset OWLS-TC[10], placed them in a local folder and inserted them in the network using the function 'Create nodes with' from the 'File' menu that automatically creates a number of nodes equal to the number of files in the local folder. Subsequently, by selecting 'Insert' from 'Edit' menu, a dialog appears that allows to insert new nodes in the network individually, specifying the URL of the bootstrap node and the URL of the new node which must be different from those used for the other nodes in the network. The next step is the population of the network. To work with DHT the choice of the *key* is a fundamental step. Our *key* was the output value of the OWL-S profile of the selected services. The output value is extracted parsing the service profile available online. The value we have associated with the key is the URI of the OWL-S service itself. Selecting 'Populate the Network' from 'File' menu this procedure is executed in automatic way for all services contained in the local folder.

The synchronous retrieval of the value associated with the key is carried out invoking the Open Chord method $retrieve(Key)$. The result is an array of strings containing the URL of zero or more OWL-S services, depending on whether the searched key is the output of one or more services inserted in the network. Figure

[10] OWLS-TC service retrieval test collection - `http://projects.semwebcentral.org/projects/owls-tc`



```
>Created a new network: initial node oclocal://localhost:8080/nodo0

>Created 36 nodes. Total nodes in the network: 37

>Network populated
>Search results for the key BOOK:
>http://127.0.0.1/services/1.1/BookFinder.owls
>http://127.0.0.1/services/1.1/BookSearchService.owls
>http://127.0.0.1/services/1.1/publication-number_book_Portalservice.owls
>http://127.0.0.1/services/1.1/publication_book_service.owls
>http://127.0.0.1/services/1.1/AcademicBookNumberSearch.owls
>http://127.0.0.1/services/1.1/AcademicBookNumberOrISBNSearch.owls
>http://127.0.0.1/services/1.1/isbn_book_service.owls
>http://127.0.0.1/services/1.1/academic-item-number_book_service.owls
>http://127.0.0.1/services/1.1/publication-number_book_service.owls
```

Fig. 4. An example of results setting $BOOK$ as key

4 shows the results obtained for the key $BOOK$ using the 'Search. . . ' pop up opening from 'Modify' menu.

By applying methods that use lexical ontologies (e.g., WordNet[11]) you can obtain the synonyms of the key. Invoking the method $retrieve(Key)$ on synonyms, those services that do not have as output the initial key are found, providing a solution to the problem of exact matching between the searched key and the output of the service.

4 Discussion

The use of P2P networks for SWS discovery opens the way for the application of intelligent methods to satisfy the requests of Web users. The availability of semantic descriptions of services allows the realization of new scenarios in which the weight of the reasoning for the attainment of a goal moves increasingly towards software systems. In the scenario shown in Figure 5, a user queries a software agent, capable of interpreting natural language, asking him to find a service that returns a certain result (Goal). The agent uses the P2P network for the discovery of services that may be suitable to meet the demand. Since the P2P network returns the semantic descriptions of services, the agent can apply automated reasoning methods to select and compose the most appropriate services with respect to the available user inputs. If the services are described by using different ontologies, it will use semantic alignment tools[12] and approaches in the literature [4, 1] during the execution of these operations. This scenario

[11] WordNet, a lexical database for English - `http://wordnet.princeton.edu/wordnet/`

[12] Alignment API and Alignment Server - `http://alignapi.gforge.inria.fr/`

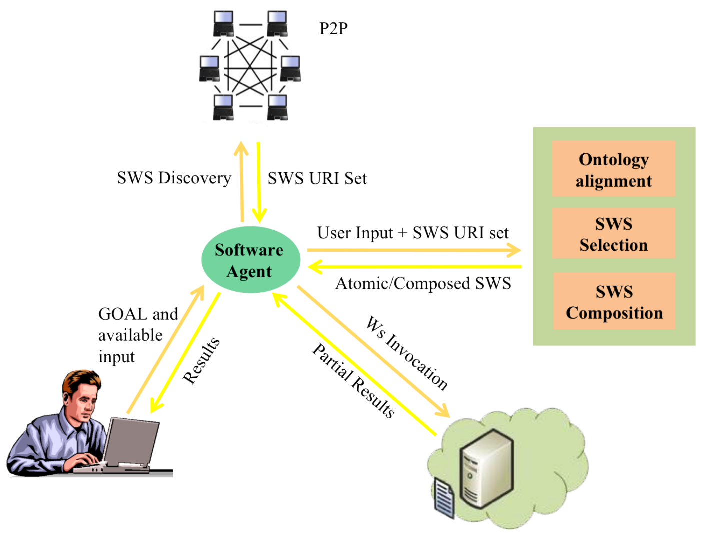
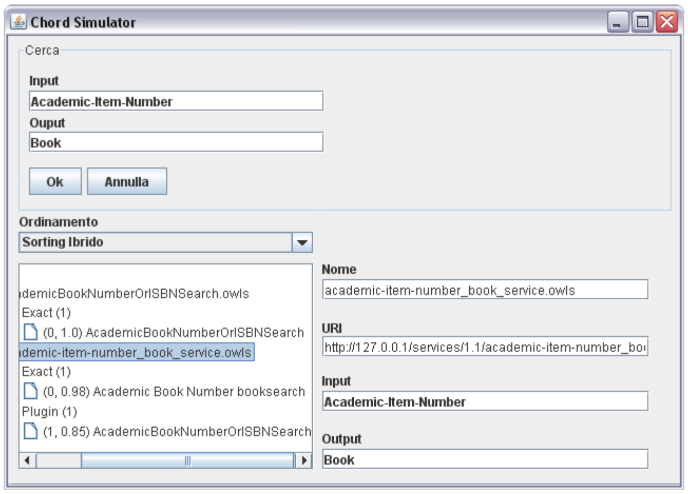


Fig. 5. Scenario

describes the automatic orchestration of SWS and is particularly suitable for use with OWL-S [12]. Looking in particular to the discovery use case, there are various matchmaking techniques that exploit the OWL-S description to determine which services are best suited to fulfil the request. Srinivasan et al. [16] propose an enhancement of CODE OWL-S IDE [15] where the used matching procedure is based on the algorithm described in [10]. It defines a flexible matching mechanism based on subsumption in Description Logics. More sophisticated solutions are provided by OWLS-MX [7], an hybrid Semantic Web Service matchmaker that retrieves services for a given query written in OWL-S itself. In other words for every OWL-S service representing the description of the desired service (query), it returns an ordered set of relevant services ranked according to their degree of (syntactic and/or semantic) matching with the query. This approach complements logic based reasoning with approximate matching relying on Information Retrieval metrics.

Figure 6 illustrates the graphical user interface (GUI) of a software component that we have developed as a support for the test of matchmaking on retrieved services. At the top there are two text fields designed to insert user inputs and the searched output of the service, respectively. By pressing the *Ok* button the request is processed. The results will vary depending on the chosen sort order (syntactic, semantic or hybrid) that reflect those available with OWLS-MX API[13]. Finally, clicking on a listed service the following information

[13] OWLS-MX Semantic Web Service Matchmaker API - `http://www.semwebcentral.org/projects/owls-mx/`



Chord Simulator

Cerca

Input: Academic-Item-Number

Ouput: Book

Ok | Annulla

Ordinamento: Sorting Ibrido

```
ıdemicBookNumberOrISBNSearch.owls
Exact (1)
  (0, 1.0) AcademicBookNumberOrISBNSearch
ıdemic-item-number_book_service.owls
Exact (1)
  (0, 0.98) Academic Book Number booksearch
Plugin (1)
  (1, 0.85) AcademicBookNumberOrISBNSearch
```

Nome: academic-item-number_book_service.owls

URI: http://127.0.0.1/services/1.1/academic-item-number_bo

Input: Academic-Item-Number

Output: Book

Fig. 6. Service discovery GUI

will be displayed: name, URI, inputs and outputs. The work presented in this paper represents only a starting point towards a SWS discovery system based solely on semantic descriptions of services. Future work includes extensive use of the annotations included in the OWL-S profile for the selection of services that best meet the user needs. For this purpose, the domain ontologies matchmaking methods will be combined with lexical ontology based approaches in order to analyze the lexical text descriptions of the service during the discovery process.

References

[1] David, J., Euzenat, J., Scharffe, F., dos Santos, C.T.: The alignment api 4.0. Semantic Web 2(1), 3–10 (2011)

[2] Doyle, J.F.: Peer-to-peer: harnessing the power of disruptive technologies. Ubiquity 2001 (May 2001)

[3] Erl, T.: Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall PTR, Upper Saddle River, NJ, USA (2005)

[4] Euzenat, J., Shvaiko, P.: Ontology matching. Springer-Verlag, Heidelberg (DE) (2007)

[5] Gharzouli, M., Boufaida, M.: Pm4sws: A p2p model for semantic web services discovery and composition. Journal of Advances in Information Technology 2(1) (2011)

[6] Karger, D.R., Lehman, E., Leighton, F.T., Panigrahy, R., Levine, M.S., Lewin, D.: Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In: STOC. pp. 654–663 (1997)

[7] Klusch, M., Fries, B., Sycara, K.: Automated semantic web service discovery with OWLS-MX. In: AAMAS '06: Proceedings of the fifth international joint conference

on Autonomous agents and multiagent systems. pp. 915–922. ACM Press, New York, NY, USA (2006)

[8] Liben-Nowell, D., Balakrishnan, H., Karger, D.R.: Observations on the dynamic evolution of peer-to-peer networks. In: Druschel, P., Kaashoek, M.F., Rowstron, A.I.T. (eds.) IPTPS. Lecture Notes in Computer Science, vol. 2429, pp. 22–33. Springer (2002)

[9] McIlraith, S.A., Son, T.C., Zeng, H.: Semantic Web Services. IEEE Intelligent Systems 16(2), 46–53 (2001)

[10] Paolucci, M., Kawamura, T., Payne, T.R., Sycara, K.P.: Semantic Matching of Web Services Capabilities. In: ISWC '02: Proceedings of the First International Semantic Web Conference on The Semantic Web. pp. 333–347. Springer-Verlag, London, UK (2002)

[11] Ratnasamy, S., Francis, P., Handley, M., Karp, R.M., Shenker, S.: A scalable content-addressable network. In: SIGCOMM. pp. 161–172 (2001)

[12] Redavid, D., Esposito, F., Iannone, L.: A comparative study on semantic web services frameworks from the dynamic orchestration perspective. In: In Proceedings of International Conference on Knowledge Engineering and Ontology Development (KEOD). pp. 355–359 (October 2010)

[13] Rowstron, A.I.T., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Guerraoui, R. (ed.) Middleware. Lecture Notes in Computer Science, vol. 2218, pp. 329–350. Springer (2001)

[14] Schmidt, C., Parashar, M.: Flexible information discovery in decentralized distributed systems. In: HPDC. pp. 226–235. IEEE Computer Society (2003)

[15] Srinivasan, N., Paolucci, M., Sycara, K.: CODE: A Development Environment for OWL-S Web services. Tech. Rep. CMU-RI-TR-05-48, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA (October 2005)

[16] Srinivasan, N., Paolucci, M., Sycara, K.: Semantic Web Service Discovery in the OWL-S IDE. In: HICSS '06: Proceedings of the 39th Annual Hawaii International Conference on System Sciences. p. 109.2. IEEE Computer Society, Washington, DC, USA (2006)

[17] Stoica, I., Morris, R., Liben-Nowell, D., Karger, D.R., Kaashoek, M.F., Dabek, F., Balakrishnan, H.: Chord: a scalable peer-to-peer lookup protocol for internet applications. IEEE/ACM Trans. Netw. 11(1), 17–32 (2003)

[18] Xu, B., Chen, D.: Semantic web services discovery in p2p environment. In: ICPP Workshops. p. 60. IEEE Computer Society (2007)

[19] Yu, S., Zhu, Q., Xia, X., Le, J.: A novel web service catalog system supporting distributed service publication and discovery. In: Ni, J., Dongarra, J. (eds.) IMSCCS (1). pp. 595–602. IEEE Computer Society (2006)