

Web Service Discovery in Large Distributed System Incorporating Semantic Annotations

Shou-jian Yu

College of Computer
Science and Technology,
University of Donghua,
Shanghai, China

Xiao-kun Ge

Shanghai Development
Center of Computer
Software Technology,
Shanghai, China

Jing-zhou Zhang

Shanghai Development
Center of Computer
Software Technology,
Shanghai, China

Guo-wen Wu

College of Computer
Science and Technology,
University of Donghua,
Shanghai, China

Abstract - *Web services are the new paradigm for distributed computing. A critical factor to the overall utility of Web services is a scalable, flexible and robust discovery mechanism. Traditional centralized indexing scheme can't scale well with a large distributed system. In this paper, we first use an ontology-based approach to capture real world knowledge for semantic service annotation. Then we use a distributed hash table (DHT) based catalog service in P2P system to index the ontology information and store the index at peers. We have discussed the DHT based service discovery model and discovery procedure. The solution to system evolution is proposed. However, DHT supports only exact match lookups, we have made improvement to the matching algorithm for flexible service discovery. The experiments show that the discovery model has good scalability and the semantic annotation can notably improve discovery exactness. The improved algorithms can discover the most potential service against request.*

Keywords: Web Service Discovery, Semantic Annotation, Distributed Hash Table

1 Introduction

Web services are emerging as a dominant paradigm for constructing distributed business applications and enabling enterprise-wide interoperability. An application can be built by integrating multiple services together making a more complex service. In order to integrate these services, one must be able to locate and acquire specified services. Existing Universal Description Discovery Integration (UDDI) [1] technology uses a central server to store information about registered Web services. However, in large distributed system, as the number of Web services grows and becomes more dynamic, such a centralized approach quickly becomes impractical. The centralized indexing scheme does not scale well, because the number

and physical distribution of the UDDI can quickly overwhelm this centralized configuration and can lead to serious performance bottlenecks. As a result, there must be some decentralized approaches to overcome these limitations.

UDDI specification does not support registering information of the service descriptions in the registry. Hence the effectiveness of UDDI is limited, even though it provides a very powerful interface for keyword and taxonomy based searching. Suggestions [2] have been made to register WSDL (Web Service Description Language) descriptions [3]. However, since WSDL descriptions are purely syntactic, registering them would only provide syntactical information about the Web services. The problem with syntactic information is that the semantics implied by the information provider are not explicit, leading to possible misinterpretation by others. Improving Web service discovery requires explicating the semantics of both the service provider and the service requestor.

In this paper, we present a methodology for building dynamic, scalable, decentralized registries with real-time and flexible search capabilities, to support Web service discovery. In our work, we focus on a fully distributed architecture motivated by recent advances in peer-to-peer computing (P2P). P2P systems research has proposed a number of new distributed architectures with desirable traits, including no central infrastructure, better utilization of distributed resources, and fault tolerance. This paper studies the feasibility of using existing P2P technology as the basis for efficient Web services discovery over an arbitrary distributed system. On the other hand, this work uses an ontology-based approach to capture real world knowledge for semantic service annotation [4]. We model the concept into ontology from two aspects, which provides the comprehensive semantics description for Web services. We index the ontology information and store the index at peers in the P2P system using a distributed hash table approach.

This work has been partially supported by the Key Programs of Science and Technology Commission Foundation of Shanghai, China under contract 05DZ11C06, and the project: Visual Document Management System for Component Development.

The rest of this paper is structured as follows: Section 2 briefly lists the related works. Section 3 presents the ontology based Web service annotation method. The detailed explanation for DHT based Web service discovery model is in section 4. In section 5, we implement a prototype. The match exactness and system scalability are evaluated. Section 6 concludes all this paper.

2 Related Work

Current approaches to Web service discovery can be broadly classified as centralized or decentralized. The centralized approach includes UDDI, where central registries are used to store Web service descriptions. Three major operators, namely IBM, Microsoft, and ARIBA provide public UDDI service. The current UDDI attempts to alleviate the disadvantages of the centralized approach by replicating the entire information and putting them on different sites. Replication, however, may temporarily improve the performance if the number of UDDI users is limited. But, the more the replication sites, the less consistent the replicated data will be.

The decentralized approaches are commonly based on P2P systems. [5] uses an ontology-based approach to organize registries, enabling semantic classification of all Web services based on domains, but maintenance of such a federated environment is a complicated thing. [6] uses a dimension reduced indexing scheme to map the multidimensional information space to physical peers, which supports complex queries containing partial keywords and wildcards. But it does not utilize the semantic description. The discovery ability is limited. Another approach is to organize peers into a hypercube graph [7]. Web service discovery is within this topology. But maintaining the hypercube with large amount of peers is inefficient.

P2P is the decentralization from traditional central model to the decentralized service-to-service model. In this model no central index is required to span the network. Particular attention has been paid to make these systems scalable to large number of nodes, avoiding shortcomings of the early P2P pioneers such as file sharing systems like Gnutella [8] and Napster [9]. Representatives of scalable location and routing protocols are CAN [10], Pastry [11], Chord [12] and Tapestry [13], henceforth referred to as Distributed Hash Tables or DHTs. [14] shows that Chord's maintenance bandwidth to handle concurrent node arrivals and departures is near optimal. Thus we choose Chord as our experimental DHT protocol.

Each of these protocols, however, allows only simple key based lookup queries. Research in the semantic web area has pointed out that annotation with metadata can help us solve the problem of inefficient keyword based searches in the current Web service architecture [15]. Semantically

described services will enable better service discovery and also allow easier interoperation among services. Several approaches have been suggested for adding semantics to Web services [16, 17]. In our approach, we use ontology to model concepts in domain specific knowledge. We model a concept in two aspects for service discovery in different level.

3 Ontology based Web Service Annotation

Current Web services standards have focused on operational and syntactic details for implementation and execution of Web services. This limits the search mechanism for Web services to keyword based searches. Research in the Semantic Web area has shown that annotation with metadata can help us solve the problem of inefficient keyword based searches. Adding semantics to Web service descriptions can be achieved by using ontologies which support shared vocabularies and domain models for use in the service description. While searching for Web services, relevant domain specific ontologies can be referred to, thus enabling semantic matching of services.

3.1 Organizing Concepts into Ontology

An ontology is a shared formalization of a conceptualization of a domain [18]. In the Semantic Web, ontologies are used to assign commonly agreed upon semantics or interpretation to particular concepts. In the following, we describe what role ontologies play in the Web services description and how they can be used to give Web service comprehensive semantic annotation.

In our approach, we identify two aspects of semantics for Web service description elicited from object-oriented model: an object can be real entity with practical meaning or an abstract entity with no real meaning. A class is the common characteristics abstraction of objects. The class has attributes and behaviors. From this point, we also model concept into domain ontology from both attributes and behaviors aspects. As shown in Fig. 1, the concept *itinerary* has attributes as *startPlace*, *arrivalPlace*, *startTime* and *No* etc. It is also associated with behaviors as *search*, *book*, *cancel* etc. The concept describes what functionality a Web service aims at. As a Web service is implemented by several functions, i.e. operations, the behaviors aspect is used to describe operations of a Web service. As an operation has inputs and outputs, the attributes aspects can be used to describe the i/o interface of operations. From the left part of Fig. 2, we can see the thorough levels of Web service architecture. By this method, Web service is semantically described in different granularity, which facilitates different service level discovery.

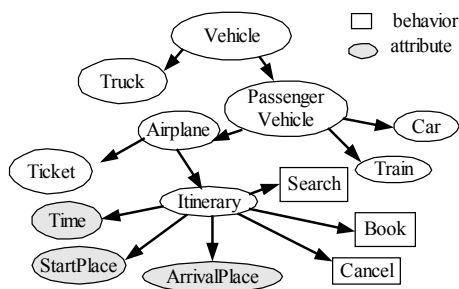


Fig. 1. Organize concepts in ontology

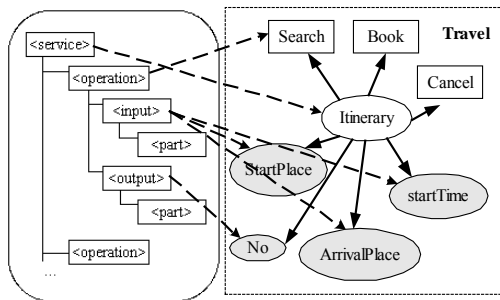


Fig. 2. Web service annotation architecture

We attempt to use minimum information to give Web service comprehensive annotations. Large distributed system can't afford much detailed information although ontology can capture much information of real world knowledge and domain theory. The detailed information can be incorporated into WSDL or UDDI. We aim at providing an efficient catalog service to improve the P2P network performance. In section 4, we will discuss how we use this semantic annotation for service discovery in large distributed system.

3.2 Annotating WSDL with Semantics

WSDL is the current standard of the description of Web service. The syntax of WSDL is defined in XML Schema. A WSDL document describes the location of a Web service, its available operations and their associated messages and data types as well as the format of their result values. The XML Schema types of message parameters may be defined using a <types> element. A <message> element is needed to compose such data types into messages. Messages need to be grouped into operations, which may define an <input>, an <output> and a <fault> message. We add semantics to Web services by mapping service, operation, input and output in their descriptions to concepts in the domain specific ontologies. The structure is shown as Fig. 2.

```
</types>
<xsd:element name="StartCity" Ont-
Concept="Itinerary:startPlace"
minOccurs="1" maxOccurs="1" type="xsd:string">
```

```
<xsd:element name="ArrivalCity" Ont-
Concept="Itinerary:arrivalPlace"
minOccurs="1" maxOccurs="1" type="xsd:string">
<xsd:element name="time" Ont-
Concept="Itinerary:startTime"
minOccurs="1" maxOccurs="1" type="xsd:string">
.....
</types>
.....
<portType>
<operation name="SearchItinerary" Ont-
Concept="Itinerary:Search">
.....
</operation>
</portType>
.....
<service name="ItineraryService" Ont-
Concept="Travel:Itinerary">
.....
</service>
```

As the above WSDL code shows, the Web service is annotated in three aspects: service *ItineraryService* is annotated by *Travel:Itinerary*, operation *SearchItinerary* is annotated by *Itinerary:Search* and i/o element *StartCity* is annotated by *Itinerary:startPlace* etc.

4 DHT Based Service Discovery

Without a central services registry, a naïve way to discovery a service in large distributed system is to send the query to each of the participant, i.e. service provider. While this approach would work for a small number of service providers, it certainly does not scale to a large distributed system. Hence, when a system incorporates thousands of nodes, a facility is needed that allows the selection of the subset of nodes that will produce results, leaving out nodes that will definitely not produce results. Such a facility implies the deployment of catalog-like functionality.

Recently a new generation of P2P systems, offering DHT functionality, has been proposed. These systems greatly improve the scalability and exact-match accuracy of P2P systems. In this section, we will discuss DHT based services discovery in large distributed system. The DHT background and Chord protocol are first introduced. Based on this, we propose the system model for semantics indexing catalog service. Then the service discovery procedure are illustrated and we give improvement to the matching algorithm for flexible discovery. At the end of this section, we discuss the system evolution maintenance.

4.1 DHT Background and Chord

DHTs are benchmarked to introduce a new generation of large-scale Internet services [19]. Their goal is to provide the efficient location of data items in a very large and dynamic distributed system without relying on any centralized infrastructure. Given a key, the corresponding data item can be efficiently located using only $O(\log n)$ network messages where n is the total number of nodes in

the system [12]. Moreover, the distributed system evolves gracefully and can scale to very large numbers of nodes. Our work leverages this functionality to provide a scalable fully distributed catalog service.

Chord, which serves as the experimental substrate of our work, is publicly available and has been successfully used in other projects such as CFS [20]. Nevertheless, our design does not depend on the specific DHT implementation and can work with any DHT protocols. There is one basic operation in the Chord systems, *lookup(key)*, which returns the identity of the node storing the corresponding data item with that key. This operation allows nodes to publish and query information based on their key. The keys are strings of digits of some length. Nodes have identifiers, taken from the same space as the keys (*i.e.*, same number of digits). Depending on the application this node is responsible for associating the key with the corresponding data item. Chord uses hashing to map both keys and node identifiers (such as IP address) onto the identifier ring. Each key is assigned to its successor node, which is the nearest node traveling the ring clockwise. Thus it allows data request to be sent obliviously of where the corresponding items are stored. Nodes and keys may be added or removed at any time, while Chord maintains efficient lookups using just $O(\log n)$ state on each node in the system. For a detailed description of Chord and its algorithms refer to [12].

4.2 System Model

Let N_i denote the n providers (a provider is a node in the identifier ring), each of which publishes a set C_i of Web services. When a node N_i wants to join the system it creates catalog information, which is the set $C_i = \{(k_j, S_{ij}) \mid S_{ij} \text{ is a summary of } k_j \text{ on node } N_i\}$.

In section 3, we have discussed the semantic annotation of Web services. We model concepts in two aspects: *behaviors* and *attributes*. Thus the concept should be the key, *i.e.* k_j , as mapped by Chord protocol and *Behavior(Attributes Set)* should be the corresponding data item associated with the key, *i.e.* S_{ij} , the summary of k_j . As a Web service consists of several operations, there are sets of data summaries S_{ij} , and not just single data summaries.

An example illustrates how the outlined conceptual catalog model can be put into practice. Consider four service providers which want to publish services as shown in Table 1. Assume that the service providers have chosen appropriate concept in the domain ontology to refer to, *i.e.* as shown in *Concept in Domain Ontology* column of Table 1. Thus this column acts as the key for mapping. Operations of the service are also semantically described by *Mapped Behavior & Attributes* column in Table 2, 3, which act as the summary for the associated key. For example, as provider N_1 : *Itinerary* is the key to be mapped.

Operation name *SearchItinerary* is referred to *Search*, which is a behavior of concept *Itinerary*, as shown in Fig. 1. (*startPlace*, *arrivalPlace*, *startTime*) is attributes of concept *Itinerary* from inputs of operation *SearchItinerary*. According to the conceptual catalog model, $S_{1, Itinerary} = \{Search(startPlace, arrivalPlace, startTime), Book(No), Cancel(No)\}$ (see Table 2), while $S_{2, Itinerary} = \{Search(No), Delivery(No)\}$ (see Table 3). Table 4 shows how the DHT assigns the keys to the nodes that joined the network. The summary sets are stored along with the keys.

Table 1. Nodes with example services

Service Provider	Web Services	Concept in Domain Ontology
N_1	Itinerary service	Itinerary
	Insurance service	Insurance
N_2	Itinerary service	Itinerary
	Car Rental service	Car
	Hotel service	Hotel
N_3	Car Rental Service	Car
	Hotel Service	Hotel
N_4	Itinerary service	Itinerary
	Hotel service	Hotel

Table 2. Operations and Mapped Behavior & Attributes for N_1 's Itinerary Service

Operation	Mapped Behavior & Attributes
SearchItinerary (StartCity, ArrivalCity, time)	Search (startPlace, arrivalPlace, startTime)
Book (ItineraryNo)	Book (No)
Cancel (ItineraryNo)	Cancel (No)

Table 3. Operations and Mapped Behavior & Attributes for N_2 's Itinerary Service

Operation	Mapped Behavior & Attributes
SearchItinerary (ItineraryNo)	Search (No)
Delivery (Ticket)	Delivery (No)

Table 4. DHT index example in each node

DHT Index	
N_1	Hotel $\{(S_2, Hotel), (S_3, Hotel), (S_4, Hotel)\}$
N_2	Itinerary: $\{(S_1, Itinerary), (S_2, Itinerary), (S_3, Itinerary)\}$
N_3	--
N_4	Car $\{(S_2, Car), (S_3, Car)\}$; Insurance $\{(S_1, Insurance)\}$

4.3 Web Service Discovery

For flexible and exact service discovery, we annotate Web service in two levels: service level by concept in domain ontology and operation level by behaviors and attributes aspects of the concept. Thus for a service discovery, the requestor should first choose appropriate concept to which the preferred service may refer. Then he (or she) decides the interested behaviors and attributes he would like to provide as input for the preferred Web service. As an example, a service requestor wants to inquire about the itinerary information from Shanghai to Beijing on 1 October 2004. Considering the above ontology, the requestor may choose *Itinerary* as the domain concept, *search* as behavior and *startPlace*, *arrivalPlace*,

startTime as attributes. This is the first step for service discovery.

We assume that the service request is submitted on N_3 . *Itinerary* serves as the DHT lookup key and it returns that summary of *Itinerary* lies in N_2 , which stores the portion of the catalog that contains *Itinerary* information. Then the request with domain ontology information is sent to N_2 . This is the second step (see Fig. 3).

On N_2 , the behaviors and attributes of the query is matched against $S_{i, Itinerary}$, ($1 < i < 3$). N_2 replies to N_3 with the node set $\{N_1\}$ since only $S_{1, Itinerary}$ matches the given query and so N_1 is the only node that satisfies the request. This is the third step.

Finally, N_3 contacts N_1 for detail information, e.g. WSDL. This example illustrates the procedure of how to use DHTs by appropriately defining k_j and S_{ij} for Web service discovery.

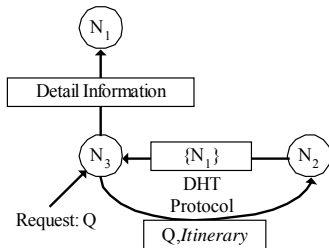


Fig. 3. Service Discovery Procedure

4.4 Improvements for Flexible Discovery

During the service discovery procedure in section 4.3, step 3 executes exact match between behaviors and attributes against summary $S_{i, Itinerary}$. Sometimes the exact match becomes inefficient because the service provider and service requestor may use approximate but not identical ontology information to express the same meaning. Considering the above example, if the requestor uses *Lookup* as behavior, he (or she) wouldn't get the result he wants. In fact, *Lookup* is a synonym of *Search*.

In our approach, we extend the exact match by using the measure of the linguistic similarity between two concepts based on their names. We use various name and string matching algorithms like NGram, synonym matching, abbreviation expansion, stemming, tokenization etc. The *NGram* algorithm calculates the similarity by considering the number of qgrams that the names of two concepts have in common [21]. The *CheckSynonym* algorithm uses WordNet to find synonyms [22]. A custom abbreviation dictionary is used for the *CheckAbbreviations* algorithm. The *TokenMatcher* uses the Porter Stemmer algorithm, tokenization, and substring matching techniques to find the similarity [23]. If any of these algorithms return a full match, we believe the match is successful. This linguistic

similarity match approach discovers the potential match between request and summary, which also results in unexpected results in some degree. Therefore we will first try to get the exact match. These algorithms won't be utilized unless no exact match results returns.

4.5 System Evolution

When a node joins/leaves the system, the affected data structure on some existing nodes must be updated accordingly to reflect the change. This section describes how the distributed catalog service evolves when nodes join, leave and update their data, and how objects, which are the sets of data summaries, are stored and accessed.

Nodes Joining

Each new node N_n (service provider) that joins the system creates the set C_n which makes its services queryable by the nodes already in the system. First N_n contacts any node N_c in the system (Step 1, Fig. 4). Node and key have the same string space. N_n gets a position in the identifier ring by the same Hash algorithm. Chord finds N_n 's successor N_s in the identifier ring. The new node N_n , now part of the identifier ring, injects each $(k_{ni}, S_{ni}) \in C_n$ into the system and the Chord protocol decides which nodes should receive the new catalog information (Step 2). Additionally, N_n becomes part of the catalog infrastructure and should share the load of the catalog service by hosting parts of the summary sets already in the network. The Chord protocol will assign to N_n keys k_i for which N_n is the successor in the identifier ring. Therefore when N_n joins, k_1 , k_2 and their corresponding summaries should be reallocated from N_s to N_n (Step 3), as shown in Fig. 4.

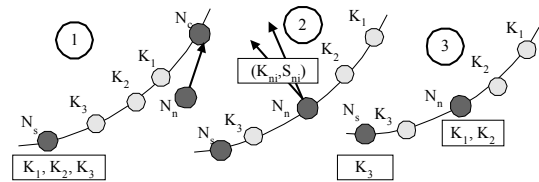


Fig. 4. Service provider joining the system

Updates and Departures

Catalog information stored in the system may need to be updated as the provider makes changes to their services. Update requests are handled in a similar way as insertion of information during join (Step 2, Fig. 4). Only the node that has created the data summary (the *owner*) is allowed to change its content. Nodes that store the data summaries are not allowed to alter their content, although they may alter the way they are stored. When a node N decides to leave the system, it must hand over the catalog information to its successor according to the Chord protocol. Furthermore, it notifies the nodes that hold N 's catalog data. To achieve this, N uses the keys it inserted into the system to find the nodes that currently hold N 's catalog information.

5 Implementation and Experiments

We have implemented a prototype to illustrate the discovery model we have proposed. The prototype is based on the Chord protocol implementation found on the Chord project website which is linked as a library. WordNet 2.0, an on-line lexical database for the English language, is embedded into the system through APIs to further understand the semantics of Web services described. As the overlay network configuration and operations are based on Chord [12], its maintenance costs are of the same order as in Chord. An evaluation of the Web service discovery exactness and the system scalability is presented below.

5.1 Web Service Discovery Exactness Evaluation

To test service discovery exactness we first obtain a corpus of Web services from SALCentral.org and XMethods.com. We have limited our testing to two domains, i.e. Weather and Geographical domains, due to lack of relevant domain specific ontologies. We compare the discovery results in three circumstances:

- The exact word match is used for discovery and the Web service is not annotated with ontology information. The service names are directly used as the key hashed to catalog indexes.
- The Web services are manually annotated with domain ontology.
- The optimization algorithms in section 4.4 are used to achieve versatile discovery.

Table 5 Web service discovery exactness evaluation results

	Domain 1		Domain 2	
	Correct Rate	Error Rate	Correct Rate	Error Rate
(a)	20%	0	17%	0
(b)	70%	0	73%	0
(c)	98%	2%	95%	2%

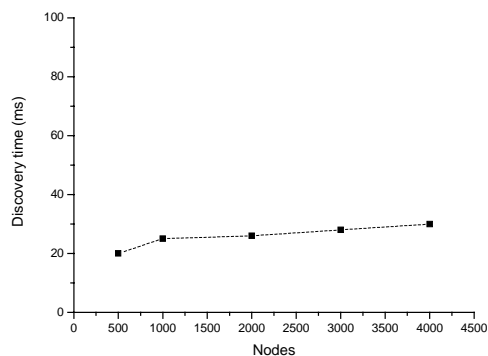
From the results in Table 5, we can conclude that method (a) has low correct rate and error rate. Comparatively, method (b) gets considerably increase in correct rate, but only about 70% are discovered. Method (c) gets almost all the services satisfying the request. But on the other hand, the error rate is a little high than the other two methods. This can be overcome by using appropriate domain ontology for Web service annotation.

5.2 System Scalability Evaluation

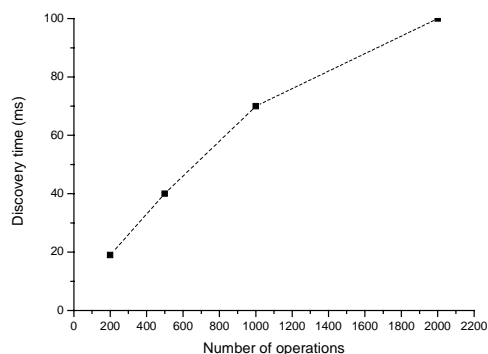
In this section, we will evaluate the system scalability by constantly changing the number of nodes and operations in Web service discovery.

From (a) in Fig. 5, we can see that the discovery time changes steadily with peer nodes increase. This is because the discovery time mainly results from the matching of query against the summary in the catalog index. Thus it is

not obviously irrelevant to the number of nodes. On the other hand, with the increasing of number of operations, the catalog index becomes larger, which results in longer discovery time as show in (b) of Fig. 5. Because the matching is processed only in one node, the increase ration of discovery time is not obvious. It inclines to stabilization. Thus this is not a critical problem for application in large distributed system. Also this can be overcome by improving the matching algorithm.



(a)



(b)

Fig. 5. System scalability evaluation results

6 Conclusions and Future Work

This paper presents a flexible Web service discovery architecture by combining semantic Web service with P2P networks. This system does not need a central registry for Web service discovery. We use an ontology-based approach to capture real world knowledge for semantic service annotation. A DHT based catalog service is used to store the semantic indexes for direct and flexible service discovery. We have discussed the discovery system model and proposed several improvements algorithms for flexible service discovery. Our experiments have shown that the semantic annotation approach suggested in this paper will significantly improve Web services discovery exactness and the DHT based discovery model has good scalability in large distribute system. With Web services being as the enabling technology for next generation network, we

believe that this service discovery infrastructure will help organizations and businesses in carrying out their business goals in a more scalable environment.

7 References

- [1] UDDI. UDDI white papers. <http://www.uddi.org/whitepapers.html>
- [2] F. Curbera, D. Ehnebuske, D. Rogers: Using WSDL in a UDDI Registry, Version 1.07, UDDI Best Practice, May 21, 2002. <http://www.uddi.org/pubs/wsdlbestpractices-V1.07-Open-20020521.pdf>
- [3] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana: Web Services Description Language (WSDL) 1.1. <http://www.w3.org/TR/wsdl>, March 2001
- [4] T. Berners-Lee, J. Hendler and O. Lassila: The semantic web. *Scientific American*, 284(5) 34–43, 2001
- [5] Kunal Verma, Kaarthik Sivashanmugam, Amit Sheth, Abhijit Patil, Swapna Oundhakar and John Miller: A Scalable Infrastructure of Registries for Semantic Publication and Discovery of Web Service. *Journal of Information Technology and Management*, 2003
- [6] C. Schmidt, M. Parashar: A Peer to Peer Approach to Web Service Discovery. *Proceedings of the 2003 International Conference on Web Service (ICWS '03)*, June 2003
- [7] M. Schlosser, M. Sintek, S. Decker and W. Nejdl: A Scalable and Ontology-Based P2P Infrastructure for Semantic Web Services. *Second International Conference on Peer-to-Peer Computing (P2P'02)*, September 05-07, Linköping, Sweden 2002
- [8] Gnutella Resources. <http://gnutella.wego.com/>
- [9] Napster. <http://www.napster.com>
- [10] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Schenker: A Scalable Content-Addressable Network. *Proc. of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*
- [11] A. Rowstron, P. Druschel: Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. *IFIP/ACM Intl, Conference on Distributed Systems Platforms*
- [12] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, H. Balakrishnan: Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *Proc. SIGCOMM 2001*
- [13] B. Y. Zhao, J. Kubiatowicz, A. Joseph: Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. UCB Tech. Report UCB/CSD-01-1141
- [14] David Liben-Nowell, Hari Balakrishnan, David Karger: Observations on the Dynamic Evolution of Peer-to-Peer Networks. *Proceedings for the 1st International Workshop on Peer-to-Peer Systems*, March 2002
- [15] McIlraith, S., Son, T.C. and Zeng, H.: Semantic Web Services. *IEEE Intelligent Systems, Special Issue on the Semantic Web*. 16(2) 46-53, March/April, 2001
- [16] Rama Akkiraju, Richard Goodwin, Prashant Doshi, Sascha Roeder: A Method for Semantically Enhancing the Service Discovery Capabilities of UDDI. *Proceeding of IJCAI-03 Workshop on Information Integration on the Web, IIWeb-03*, August 2003
- [17] DAML-S Coalition: DAML-S: Web Service Description for the Semantic Web. *ISWC01*, 2002
- [18] M. Uschold and M. Grüninger: *Ontologies: Principles, Methods and Applications*. In *Knowledge Engineering Review*, 11(2), 1996
- [19] H. Balakrishnan, F. Kaashoek, D. Karger, R. Morris, I. Stoica: Looking up data in P2P systems. *Communications of the ACM*, Feb 2003
- [20] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, I. Stoica: Wide-area cooperative storage with CFS. *SOSP 2001*
- [21] R. C. Angell, G. E. Freund, et al: Automatic spelling correction using a trigram similarity measure. *Information Processing and Management* 19(4) 255-161, 1983
- [22] G. Miller: Special Issue, WordNet: An on-line lexical database. *International Journal of Lexicography*, Vol. 3, Num. 4, 1990
- [23] M. Porter: An algorithm for Suffix Stripping. *Program – Automated Library and Information Systems*, 14(3), 1980