Ontology-Based Service Discovery Front-End Interface for GloServ

Knarig Arabshian[1], Christian Dickmann[2], and Henning Schulzrinne[3,⋆]

[1] Alcatel-Lucent Bell Labs
[2] VMWare
[3] Dept of Computer Science, Columbia University

Abstract. This paper describes an ontology-based service discovery front-end interface for GloServ. GloServ is a service discovery engine, which is an ontology-based distributed service discovery system that allows sophisticated querying of services. The working implementation of the front-end interface demonstrates how GloServ can be used for different types of web service discovery. The front-end generates a search form from the service class ontology. It also allows multiple services to be queried for in a single search by generating cascaded forms for combined service queries. It then converts the input to a GloServ query and displays the results to the user in a coherent manner. The use cases that are demonstrated with this implementation are service discovery for location-based services, tagged services and collaborative search with other users.

Keywords: User interface, service discovery, ontologies, OWL, CAN, peer-to-peer.

1 Introduction

This paper describes an ontology-based service discovery front-end user interface to GloServ [12] [13] [11]. GloServ is an ontology-based global service discovery system. It uses the Web Ontology Language Description Logic (OWL DL) [5] to classify services in an ontology and map knowledge obtained by the ontology onto a hybrid hierarchical peer-to-peer network. It operates in wide as well as local area networks and supports a large range of services that are aggregated and classified in ontologies. A partial list of these services include: events-based, physical location-based, communication, e-commerce or web services.

GloServ provides a generic back-end service discovery framework for different front-end systems to interact with it. The contribution of this paper is a working implementation of a web-based front-end which demonstrates how GloServ can be used for different types of web service discovery. The use cases that are demonstrated with this implementation are service discovery for location-based services, tagged services and collaborative search with other users.

⋆ This work was developed at the Dept of Computer Science, Columbia University.

L. Aroyo et al. (Eds.): ESWC 2009, LNCS 5554, pp. 684–696, 2009.
© Springer-Verlag Berlin Heidelberg 2009

In order to build a web-based front-end to GloServ, the following problems need to be addressed: 1) downloading the service class ontologies from the GloServ back-end; 2) generating a search form from the service class ontology files; 3) allowing multiple services to be queried for in a single search by generating cascaded forms for combined service queries; 4) converting the user input to a GloServ query; 5) displaying the results to the user in a coherent manner.

Below, we describe the solution of the design and implementation of the web-based front-end system. We begin by giving an overview of the back-end GloServ service discovery system in Section 2. In Section 3 we motivate the need for a web-based front-end by describing a few use cases. Sections 4 and 5 describe the front-end system and its implementation, respectively. Finally we discuss related work in service discovery in Section 6 and conclude in Section 7.

2 Overview of GloServ

This section gives a brief overview of GloServ, the back-end service discovery architecture. For further details on the design, implementation and evaluation of GloServ as well as related work to service discover, we encourage the reader to refer to [12] [13] [11].

One of the main components of GloServ is the service classification ontology. At a high-level, services are classified in a pure hierarchy where each service class is disjoint from the other. Hence, service instances will only be classified within one of the branches. At the lower levels of the ontology, classes may have relationships with other classes and a pure hierarchy is not maintained. The upper hierarchical ontology which defines high-level services is mapped onto a hierarchical network and the low-level ontologies are mapped to a peer-to-peer network.

Another component of GloServ is the back-end hybrid hierarchical peer-to-peer service discovery network. The high-level hierarchical ontology maps to a physical hierarchical network. Each service class ontology maps to the Content Addressable Network (CAN) [16] which is a peer-to-peer network architecture. Since GloServ achieves load distribution, fast query and update processing time, while maintaining reliability, we have elected this as the underlying service discovery system.

GloServers maintain three types of information: a service classification ontology, a thesaurus ontology and, if part of a peer-to-peer network, a CAN lookup table. The high-level service classification ontology is not prone to frequent changes and thus can be distributed and cached across the GloServ hierarchical network. Each high-level service will have a set of properties that are inherited by all of its children. As the subclasses are constructed, the properties become specific to the particular service type. The thesaurus ontology maps synonymous terms of each service to the actual service term within the system. Figure 1 gives an overview of the GloServ architecture generated from the *Restaurant* ontology and how queries are routed to the correct servers.

At the lower levels, maintaining a purely hierarchical ontology structure becomes difficult as classes tend to overlap. Thus, in order to efficiently distribute service instances according to similar content, servers that hold information on

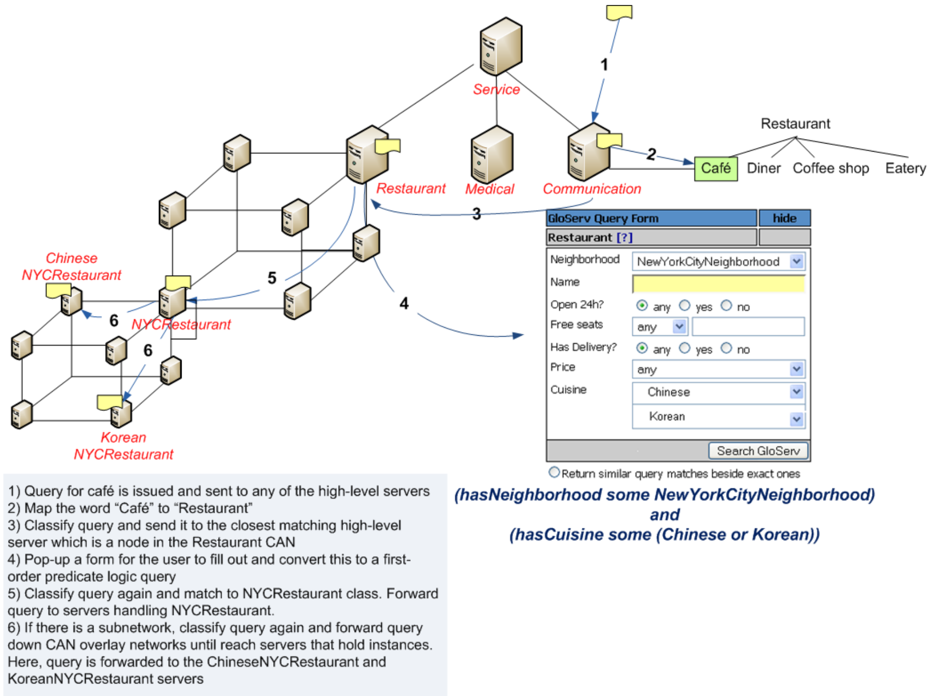

Fig. 1. Finding servers in GloServ

similar classes are distributed in a peer-to-peer network. The CAN peer-to-peer architecture distributes classes with similar content and is generated as a network of n-level overlays, where n is the number of subclasses nested within the main class. The first CAN overlay is a d-dimensional network which has the first level of subclasses of the *Restaurant* class. The number of dimensions is determined by the maximum number of nodes which can be added into the CAN. This is estimated to be $(log_2 n)/2$ to ensure that the number of query hops are $O(log_2 n)$. Services are represented as instances of the service classes and reside in the CAN servers.

A user initially contacts a GloServ user agent and enters a service name. The initial GloServer is found after following the steps outlined in Figure 1. Since each hierarchical node handles a class which is disjoint from its siblings, the query is routed down only one branch reducing the query hops considerably. Once the correct GloServer is contacted, the user agent obtains the ontology pertaining to that service class. The interface to the user can either be human-centric or automated, depending on the implementation. In either case, a query is formed and sent to the GloServer. The query is a first order predicate logic statement that contains restrictions on various properties such as: (*hasLocation* **some** `NYC`) and (*hasCuisine* **some** (`Korean` **or** `Chinese`)). The server handling that service class creates a class with this query restriction and classifies it in its

ontology. Since the subclasses of the *Restaurant* class are restricted by location, the query class gets classified as a subclass of the *NYCRestaurant* class. The query is then forwarded to the nodes that handle *NYCRestaurant* classes. When a node is found, the query class is classified again. Since the *NYCRestaurant* class has subclasses that have cuisine restrictions, when the reasoner classifies the query class, it becomes a subclass of *NYCRestaurant* and a superclass of *KoreanNYCRestaurant* and *ChineseNYCRestaurant* classes. The classification indicates that the query must be routed to the servers handling Chinese and Korean restaurants. In order to route the query within a CAN, the query needs to reduce to a dimension and key. The dimension and key values assigned to each of these classes are used during the CAN network generation to convert the ontology class to a $< dimension, key >$ pair.

GloServ allows ontology-based querying with key word matching. Since the service ontology may not capture all parts of a particular service, we allow each ontology to have a keyWords property which service providers can populate with keyword terms that describe the specifics of their service. The ontology-based query is executed and the remaining results are refined by matching the keywords entered. Gloserv also allows a combination of queries across different service classes. Thus, one can search for services with a common property, such as location, in a single search. An example of this is querying for a restaurant and a nearby movie theater.

3 Use Cases

The motivation to create a web-based front-end to GloServ is to demonstrate that GloServ can be easily plugged into a Web Service Architecture [8] and perform interesting service discovery searches. The Web Service Architecture may have different types of service discovery systems such as centralized registries as in UDDI [10], text-based search engines as in Google or peer-to-peer architectures such as GloServ. Using GloServ for web service discovery shows that service discovery can be done using a richer description language as well as within a distributed architecture. We demonstrate that a web service front-end can be implemented which discovers services using GloServ. However, we have not built a front-end which interacts with the service providers directly using SOAP and HTTP as this is not applicable to the service discovery phase and is beyond the scope of this paper. Below, we describe three use cases that a web-based front-end to GloServ can be used for: location-based service discovery, searching for tagged services and collaboratively searching for services.

3.1 Location-Based Services

Consider location-based services such as restaurants, theaters, and traffic. Imagine a couple is planning an evening out in New York City and would like to find a seafood restaurant near the waterfront in Manhattan that also has a theater nearby playing an action movie. In order to find the quickest route to the

restaurant, the closest restaurant which has the least congested route needs to be found. It should also be located near a movie theater which is featuring action movies with showtimes that are some time after dinner is over.

To perform the above search, location data must be integrated across the different domains. First, seafood restaurants are searched for in New York City which have an additional attribute of being located near a waterfront. The location of these restaurants are then fed into a traffic service network which returns the best route to the restaurant. The locations of these restaurants are chosen and then fed into the theater service network along with attributes for movie and showtimes. The results retrieved will be all the restaurants and nearby theaters which are located in places that offer the least congested route.

Currently, in order to perform such a search, a user must issue multiple queries to restaurant, theater and traffic sites in order to determine the best restaurant and theater to go to. Since data for each of these services is represented in different formats, there is no way to integrate the data while composing services. The use of ontologies would greatly aid in performing these types of complicated queries. Furthermore, if we take into account all types of location-based services, global distribution of data becomes necessary. Thus, we have chosen to use GloServ as the underlying service discovery system since it provides an ontology-based service search in addition to global distribution of services.

In this use case, three service classes are queried: *Restaurant*, *Theater*, and *Traffic Reports*. These classes all share a common property, namely, location. Since an ontology is useful in mapping semantically equivalent concepts to one another a *Location* ontology can be defined and shared across all the different service domains. The front-end downloads ontologies for the multiple service classes by providing links to the service forms. It then exhibits three cascading forms that appear to the user, one for each service class with the shared location field appearing once. The results are displayed in a Google map.

3.2 Service Tagging

Another example of a use case for a web-based front-end is searching for services and their tags. Currently, tagging is becoming a common phenomenon in various web services such as Flickr [3] and de.licio.us [2]. We define a *tag* as an attribute of a service that is not part of the service description itself. For example, many users like to read and give feedback pertaining to services they will use or have used. This review system can be deemed as a *Tag* service class which has any number of different rating services such as Zagat [9] for restaurants, Better Business Bureau for businesses, or regular user reviews. Given this type of service composition, one can search for any service which has a particular rating and give feedback for this service as well. A *Tag* service class is defined with its own service classification ontology and inserted into the GloServ network.

The relationships between GloServ service classes and their corresponding tag services can be determined using an ontology mapper. For example, tags from the *Zagat* and *NY Times* classes match the *Restaurant* service class for restaurant ratings and the *NY Times* class also matches the *Theater* class for movie ratings.

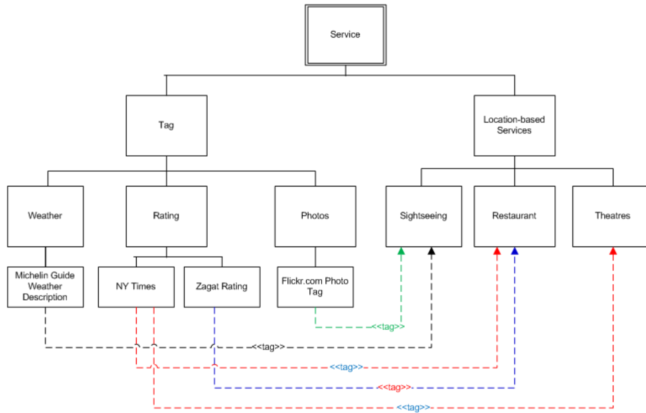

Fig. 2. Relationship ontology of the Tag class and the Location-based Services class

Flickr and *Weather* tag classes match the *SightSeeing* service class because each of these can tag various travel sites such as travel brochures or tourist spots. The front-end establishes the relationship table of a given service class and its corresponding tagging service by using an ontology mapper.

Figure 2 shows the relationships between *Tag* services and *Location-based Services.* Front-end web services handling sightseeing or restaurant services know about the tagging providers that contribute tags to their services via this relationship ontology. In most cases, these tags will be contributed by third party entities.

Searching for restaurants and theaters with specific reviews becomes simple with the front-end we have designed. The user queries for *Restaurant* and *Rating* by choosing these service links and entering data into the restaurant and rating forms. Users can also add tags to a service by providing the service URN and the reviews in the rating form.

3.3 Collaborative Search

An interesting application of GloServ is letting users collaborate in a single search. Imagine two mobile users who decide on a last minute dinner meeting. They would like to perform the search together but one of them is sitting in a meeting and is unable to talk. Thus, her friend invites her to collaborate with her on a service discovery search through GloServ. The invitation is sent via email or text message which includes a link to the GloServ collaborative interface. As *user A* clicks on her preferences, the values change on *user B*'s front-end as well, by synchronizing the interfaces. The values entered by both users is converted to a regular GloServ query and issued to GloServ.

4 Web-Based Front-End

The front-end is a web server that runs Apache and PHP [6] (version 5.1.x). It allows users to register and query for location-based services. Currently, the

service classes that are supported are for *Restaurant*, *Theater*, *Weather* and *Tagging* services. The interface provides links for each service class. When the user clicks on a service class name, a form is generated for that service class. The query results are displayed in a Google map as well as in a list. The overall GloServ front-end can be seen in Figure 3. This section describes how the front-end operates in order to perform location-based service discovery using GloServ.

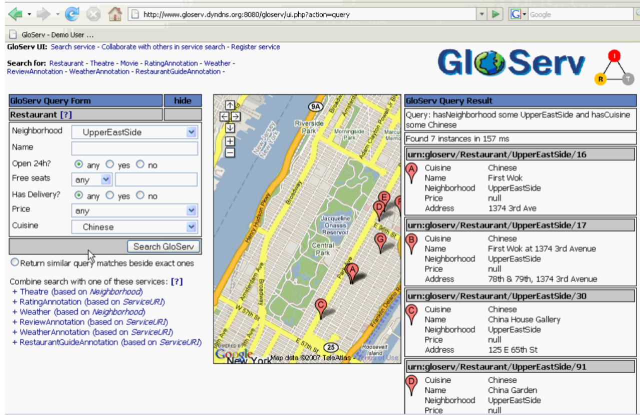

Fig. 3. GloServ Front-End

4.1 Generating a Search Form

The first step that a front-end needs to accomplish is to download the correct service ontology files from the GloServ back-end. Depending on what the web service is, it contacts the GloServ back-end by submitting the service class name. This query is routed through the GloServ hierarchy as described in Section 2 and the ontology is returned to the front-end server. The front-end caches these ontology files as well as the hostnames for the CAN super nodes of the service classes and refreshes these periodically. In this way, the query does not have to route through the hierarchical servers but can go directly to the CAN level.

The front-end parses the ontology and displays each property as a field in the form. Each property is annotated with a label property which is used for the graphical user interface. The ontology parser parses out each property's label and displays it as the field's label in the form. The form field is determined by the property's range. If the range is an object property, all the classes in the range are parsed and displayed as a drop-down menu. For datatype properties, the field is a regular text box. For example, the hasCuisine property has annotation properties label.us=`Cuisine` and label.de=`Kueche`. These two labels signify how to display the hasCuisine field in the English and German languages. The form field is a drop down menu of the *Cuisine* class and displays all the subclasses of

Fig. 4. Ontology Form

Cuisine. Similarly, the `hasName` datatype property is labeled `Name` and its field is a text box. Figure 4 illustrates this concept.

4.2 Cascading Forms for Combined Service Queries

We have built a front-end that supports combined querying of services. In order to accomplish this, the user interface needs to be able to display more than one service form at a time. The front-end server downloads a number of ontologies for each service class. In order to allow multiple services to be searched for in a single query, the relationship between these service classes needs to be established. This is accomplished by passing the ontologies through an ontology mapper which establishes the relationship between the ontologies. We implement a simple ontology mapping tool by storing the relationship between services in a relationship table. This also includes the corresponding matching properties.

The front-end interface displays the combined service classes as links. When users click on these links, cascading forms are displayed. The matching property or properties are only displayed once and inserted as a shared property in the combined query. Figure 5 below shows a combination of *Restaurant* and *Theater* service forms displayed with one *Neighborhood* field which is the common property.

GloServ Query Form hide
Restaurant [?]
Neighborhood any
Name
Open 24h? any yes no
Free seats any
Has Delivery? any yes no
Price any
Cuisine any
Theatre [?] remove
DressCode any
Name
Genre any
Director
Year any
Country any
Title
Imdb ID
Search GloServ

Fig. 5. Cascading Service Forms

4.3 Creating GloServ Queries

Single service ontology queries in GloServ are a conjunction of all the property values. Although more complicated queries can be formed (ie. disjunctive queries, cardinality assignments, and set equivalences), a conjunctive query suffices for the purpose of the demonstrative use cases. Besides creating simple ontology queries, the front-end also checks to see if there are combined queries or keywords entered and creates the query accordingly. The information entered in by the user is converted to a GloServ query. For example, `hasCuisine` **some** `Italian`, signifies that the object property hasCuisine is assigned to `Italian`.

For combined queries, the front-end creates a *primary* query out of the first service class and *nested* queries from subsequent service classes. In the example above, the *Restaurant* service query will be the primary one and the *Theater* query, the nested one. If there were more services cascaded in the form then these would be nested within each other.

For the collaborative user search, a single query is formed given inputs from multiple users. The interface is synchronized across user group. Every time a user enters a value for a property, her collaborators see an updated form with the property value the user entered. Once every user inputs a value for a given property, the query that is constructed is a conjunction of each property value entered by the users.

The query is put in an XML message which indicates what type of query it is (service registration or query). It is then passed onto the front-end server which constructs the appropriate GloServ message and issues the query to the GloServ backend.

4.4 Displaying Results

For single service query results, the results are displayed in a list where each instance and its properties are displayed. In combined queries, results are displayed using an instance tree that shows the relationship between each service instance. An example of this is the combined query of *Restaurant* and *Theater*, where the tree is formed such that instances of the *Restaurant* class are parents of matching *Theater*.

For location-based services, the results are also displayed in a Google map. Each service instance is labeled with a pin on the Google map. For combined service queries, the map shows the primary query tagged in red and the nested query tagged in green. When clicking on either the red or green tags, the user can choose to show only corresponding services. This grays out all the services

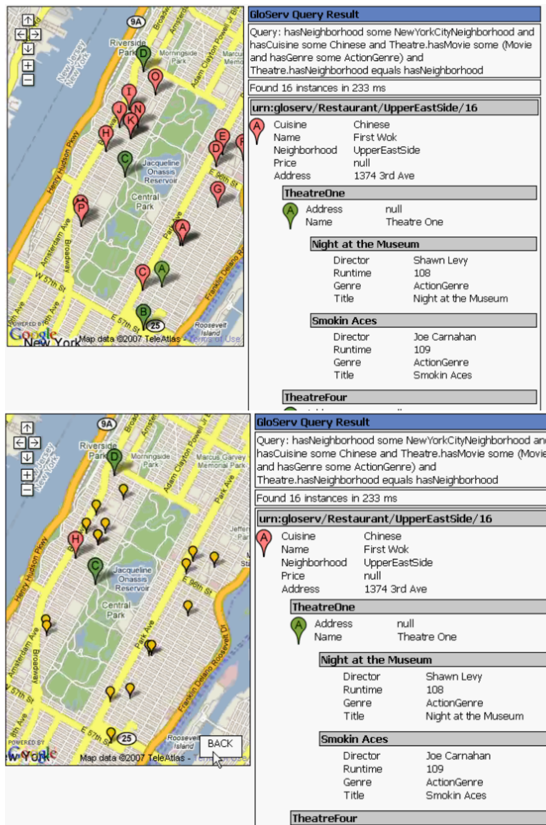

Fig. 6. Google map results for combined service queries

except for those services which match the current one that has been clicked. In this way, users can easily navigate through the map and can determine what the matching services are. Figure 6 shows how the query results are displayed for a combined restaurant and movie search.

5 Implementation

The front-end implementation is done mostly in PHP. For the collaborative interface, functionality in JavaScript [4] has also been defined. The front-end uses a Smarty template engine to generate HTML code for PHP [7]. Smarty facilitates a manageable way to separate application logic and content from its presentation. The collaborative search interface requires user view synchronization and a central data storage. The Asynchronous JavaScript and XML (AJAX) [1] framework is used to communicate with the client side and web server. AJAX provides a framework for creating efficient and interactive web applications.

GloServ nodes communicate with each other using a form of property-value-pair encoding. The implementation separates the message format and encoding from the logic. Figure 7 below shows how the messages are exchanged between the user's browser, the front-end server, the Smarty template engine and the GloServ back-end interface. It is assumed that the front-end server already has the ontology file in the cache.

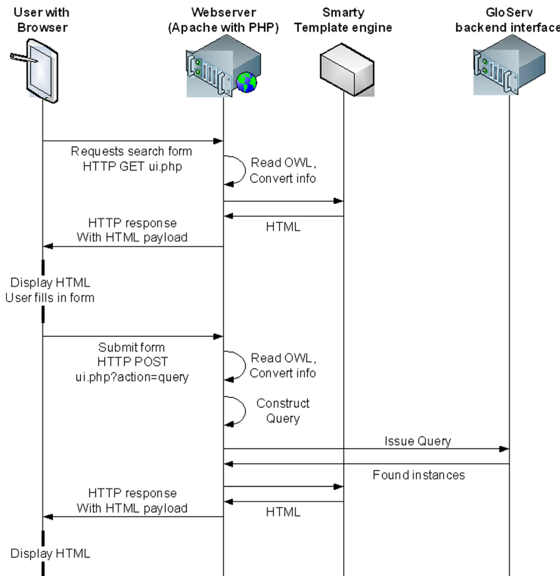

Fig. 7. Query Code Walk

6 Related Work

There are a few service discovery protocols in use today. Most service discovery mechanisms are localized and use attribute-value pairs for service descriptions. Below we describe each of these and compare them to GloServ.

SLP [14] and Jini [15] are both similar in that they have agents that manage services, users and directories of services. Agents advertise each others' presence to each other using either multicast or unicast. In SLP, service registration and queries are broadcast to the directory agents or directly between the service and user agents depending on if the directory agents are present. In Jini, however, a client downloads the service proxy and invokes through Java RMI in order to access the service through a discovery process. Service descriptions in SLP are done in simple attribute-value pairs whereas Jini matches interfaces. SLP is mainly used in local area networks. Jini can span to a larger enterprise networks. The Universal Description, Discovery and Integration (UDDI) [10] specification is used to build discovery services on the Internet. UDDI provides a consistent publishing interface and allows programmatic discovery of services. Services are described in XML and published using a Publisher's API. Consumers access services by using the Programmer's API built on top of SOAP. Services in UDDI are stored in a centralized business registry. The main drawback of UDDI is that it has a centralized architecture and does not span to a global area.

Since these architectures are either centralized or scale to a local level, GloServ provides the best architecture for service composition in a globally distributed network. Also, because of its use of ontologies, a friendlier user interface can be implemented which allows intelligent querying of services.

7 Conclusion

GloServ provides ontology service descriptions as well as a framework for different service classes to be aggregated in a single network. Due to these attributes, we have built a web-based front-end to demonstrate the interesting use cases for GloServ. These include searching for a combination of location-based services, tagged services as well as collaborating with other users on a single search.

References

1. Asp.net ajax, `http://www.asp.net/ajax/`
2. del.icio.us, `http://del.icio.us/`
3. Flickr, `http://www.flickr.com`
4. Javascript, `http://www.javascript.com/`
5. Owl: Web ontology language, `http://www.w3.org/2004/OWL/`
6. PHPs hypertext processor, `http://www.php.net/`
7. Smarty template engine, `http://www.smarty.net`
8. Web services architecture, `http://www.w3.org/TR/ws-arch/`
9. Zagat survey, `http://www.zagat.com/`

10. UDDI technical white paper. white paper, uddi (universal description, discovery and integration). Technical report, OASIS (September 2000)
11. Arabshian, K., Dickmann, C., Schulzrinne, H.: Service composition in an ontology-based global service discovery system. Technical report, Columbia University, New York, NY (September 2007)
12. Arabshian, K., Schulzrinne, H.: An ontology-based hierarchical peer-to-peer global service discovery system. Journal of Ubiquitous Computing and Intelligence (JUCI) 2, 133–144 (December)
13. Arabshian, K., Schulzrinne, H.: Combining ontology queries with key word search in service discovery. In: ACM/IFIP/USENIX 8th International Middleware Conference, Newport Beach, California (2007)
14. Guttman, E., Perkins, C., Veizades, J., Day, M.: Service location protocol, version 2. RFC 2608, Internet Engineering Task Force (June 1999)
15. Sun Microsystems. Jini architectural overview. Technical report (1999)
16. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content addressable network. In: Proceedings of ACM SIGCOMM 2001, San Diego, CA (2001)