WSPDS: Web Services Peer-to-peer Discovery Service[†]

Farnoush Banaei-Kashani, Ching-Chien Chen, and Cyrus Shahabi
Computer Science Department,
University of Southern California,
Los Angeles, California 90089
[banaeika,chingchc,shahabi]@usc.edu

Abstract— *The Web Services infrastructure is a distributed computing environment for* service-sharing*. In this environment, resource discovery is required as a primitive functionality for users to be able to locate the services, the shared resources. A discovery service with centralized architecture, such as UDDI, restricts the scalability of this environment as it grows to the scales comparable with the size of the web itself. In addition, current extensively used web service standards (e.g. UDDI, WSDL), do not support discovery at a semantic level. In this paper, we introduce WSPDS (Web Services Peer-to-peer Discovery Service), a fully decentralized and interoperable discovery service with semantic-level matching capability. We believe the peer-to-peer architecture of the semantic-enabled WSPDS not only satisfies the design requirements for efficient and accurate discovery in distributed environments, but also is compatible with the nature of the Web Services environment as a self-organized federations of peer service-providers without any particular sponsor.*

Keywords— **Web Services discovery, Peer-to-peer discovery, Ontology, Semantic matching**

I. Introduction

The Web Services programming infrastructure is the current generation of a succession of systems proposed to develop distributed applications: RPC, CORBA, DCOM, and now Web Services. A web service is a self-contained application module with well-described functionality that can be invoked across the web. The Web Services programming environment is a distributed computing environment in which participants share their services; hence, a *service-sharing* environment. Each participant can potentially act both as a service provider and as a client. As a service provider, the participant builds and optionally shares its services for public use. As a client, on the other hand, the participant can develop distributed applications by discovery and seamless integration of the public services with its own private services.

The Web Services infrastructure is adopted more rapidly and widely as compared to its predecessors. One could anticipate popularity of this infrastructure in advance, because it is an extension of the successful browser-based web-programming technology to a general distributed application development environment. However, more importantly, success of this infrastructure must be attributed to its fundamental features:

- Loose coupling: services developed and deployed independently using heterogeneous platforms can be integrated seamlessly to build distributed applications with new functionalities; hence, interoperability. Loose coupling is mainly enabled by XML-based SOAP communication specification, which allows platform-independent information exchange between services.
- Full decentralization: all communications of the interacting entities are in a peer-to-peer fashion, without any central coordination; hence, scalability.
- Semantic level search: this feature allows web service requesters to search for published web services not only based on keywords, but also based on ontological concepts.

A. Discovery Service for Web Services

In general, in a distributed computing system a discovery service locates (or discovers) resources dispersed across the system in response to resource discovery queries issued by the system entities. With Web Services, resources are the services shared on the web. To be specific, a discovery service for Web Services is itself a web service that locates the service description document(s) of the service(s) that hit a service query. A service description document (e.g., a WSDL file) provides both abstract and concrete information required for proper invocation of a service. A service query characterizes a set of services with particular characteristics, such as name, abstract(or description), interface model, etc., to be located.

B. Design Issues and Approaches for Discovery Service

To be compatible with the fundamental features of the Web Services infrastructure (as discussed

[†] This research has been funded in part by NSF grants EEC-9529152 (IMSC ERC), IIS-0082826 (ITR), IIS-0238560 (CAREER), IIS-0324955 (ITR) and IIS-0307908, and unrestricted cash gifts from Okawa Foundation and Microsoft. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

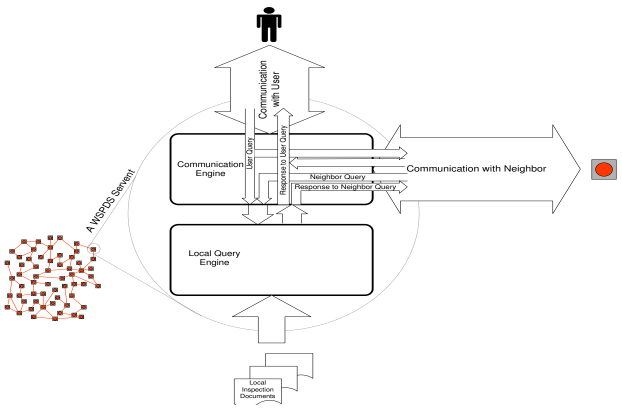


Fig. 1. WSPDS Architecture

above), a discovery service should support the following requirements:

- *Interoperablity*, to be integrable with other web services, to support different service description standards, and to be portable to different platforms;
- *Scalability*, to grow to the web scales without being a performance bottleneck;
- *Efficiency*, to support the dynamic environment of the Web Services with frequent changes/updates of the location of the services and their description documents;
- *Fault tolerance*, to be resistant to unwanted breakdowns and malicious attacks.
- *Semantic based discovery*, to find a match based on the common conceptual space of service requesters and providers.

We argue that as compared to a centralized architecture (e.g., UDDI [1], the currently used standard for globally publishing and locating web services), a decentralized design for the Web Services discovery service is more scalable (obviously), more fault tolerant (by eliminating the single point of failure), and more efficient (by reducing the overhead of centralized update of the discovery service). Distributed directory services and peer-to-peer services are two alternative service models with decentralized architecture. Distributed directory servers are usually dedicated facilities that are built and maintained under unique management to provide service to the clients of a distributed environment. However, the Web Services infrastructure is a self-organized federation of *peer* entities without any particular sponsor for the system. It is desirable that the federation lives, changes, and expands independent of any distinct service facility with global authority. With peer-to-peer services, the role of distinct service providers is eliminated. System entities all cooperate to provide a service as a result of group collaboration in a distributed fashion. Entities are peers in functionality and each entity is potentially both a server and a client of the peer-to-peer service; hence, sometimes entities are referred to as *servents* (i.e., server and client).

The Web Services discovery service can be implemented as a peer-to-peer service, eliminating dependency on a distinct service provider. Each servent serves others by providing information about its own web services in response to queries, and in turn, as a client it issues discovery queries to locate the web services that are not available locally. Servents build a network in which each servent has a few other servents as neighbors. When a servent receives a request for a web service from the local user and cannot find the web service locally, as a client it originates a discovery query and propagates the request into the network through its neighbors. Servents collaborate based on a distributed algorithm to disseminate the query. During propagation of the query, if a servent finds the requested web service locally, it responds to the originator by providing its location and description.

Additionally, in order to achieve efficient query propagation in a peer-to-peer environment, the linkage between servents should be built based on the hosted data contents (e.g., web service descriptions) of the servents. Finally, a more accurate match will be accomplished by annotating both the advertised web services and users' requests with globally shared concepts.

II. Peer-to-peer Discovery Service for Web Services

The Web Services infrastructure is a self-organized federation of service providers for service-sharing. Thus, a peer-to-peer architecture is an appropriate choice for the discovery service in this environment. Considering the usual autonomous behavior of the service providers, an unstructured peer-to-peer discovery service is preferred. Here, we introduce WSPDS (Web Services Peer-to-peer Discovery Service), a fully decentralized and interoperable discovery service with an unstructured peer-to-peer architecture.

A. Architecture

WSPDS is a distributed discovery service implemented as a cooperative service. A network of WSPDS servents collaborate to resolve discovery queries raised by their peers. Figure 1 depicts an unstructured peer-to-peer network of WSPDS servents. Each servent is composed of two engines, communication engine and local query engine, standing for the two roles that a servent plays:
1. Communication and Collaboration: the communication engine provides the interface to user and

also represents the servent in the peer-to-peer network of servents. This engine is responsible for the following tasks:

- Receiving service queries from users, resolving the queries by local query (through the local query engine) and global query (via its peer servents), and finally merging the received responses to reply to the user query; and
- Receiving queries from its neighbors in the peer-to-peer network, resolving the queries by local query, and sending the response (if not empty) to the network as well as forwarding the query (if query has still some time to live, i.e., TTL $>$ 0) to other neighbors in the network.

2. Local query: the local query engine receives the queries from the communication engine, queries the local site (where the servent is running) for matching services, and sends responses to the communication engine.

In the following sections, we first explain the implementation of the two engines to build a primitive WSPDS network based on the basic peer-to-peer network specification Gnutella[2]. The primitive WSPDS supports only keyword-matching queries. Thereafter, we describe our approach to add ontological concepts to the primitive WSPDS to achieve semantic-based peer-to-peer network construction (termed *Sem-WSPDS*) and service discovery.

III. Construction of a Primitive Peer-to-Peer Network of WSPDS Servents

A. Communication Engine

Consider to build a peer-to-peer network of WSPDS servents based on Gnutella protocol. The communication engine of a WSPDS servent exchanges SOAP-enveloped query/response messages with 1) user applications/services, or 2) other WSPDS servents. The only difference between these two types of communications is a unique identifier and a TTL field embedded in the `MessageDescriptor` of the messages exchanged between two servents (with the second case above), for peer-to-peer collaboration purposes. Obviously, these fields are not required for the messages communicated between a user application and WSPDS servent (the first case). Figure 2 shows sample communications between a WSPDS servent and a user.

Figure 5 (see the appendix) depicts the main routine that implements the communication and message handling tasks of the communication engine. Instead, here we focus on the mechanisms implemented by the communication engines of the peer servents to 1) build and maintain the peer-to-peer network of servents, and 2) execute cooperative discovery. These mechanisms are mostly compatible with the Gnutella peer-to-peer network specification [2] enhanced by our novel technique termed *probabilistic flooding.* In [3], we prove that this technique improves scalability of Gnutella's *flooding-based* dissemination mechanism by up to 99%, effectively eliminating the major drawback of this Gnutella-like peer-to-peer discovery system.

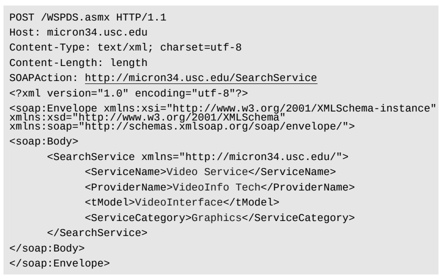
A.1 Network Setup

Each servent maintains a list of the most recently active servents of the network, denoted as *servent cache.* Each time a servent is re-activated, it probes the servents listed in the servent cache to find k nodes that are still active and designates them as its neighbors. In this way, a new servent can join the peer-to-peer network based on the local information without any unique global control. For the first time a servent is activated, the servent cache contains access points of a few WSPDS servents associated with some large service providers that are almost always active. When a servent joins the network, it periodically uses a Gnutella-like ping-pong mechanism to find other active servents in the network and refreshes its local servent cache to be updated for the next re-activation.

A.2 Cooperative Discovery

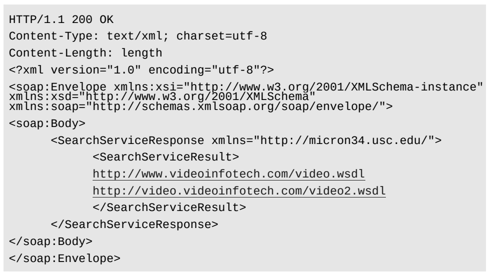
To discover a service requested by user, a servent originates a query (enveloped in a SOAP message) in the network of servents. The servents collaborate to propagate the query based on the probabilistic-flooding dissemination mechanism. Dissemination of a query is restricted by its TTL. A servent that receives a copy of the query message decreases TTL of the query by 1, and if TTL > 0, forwards the query to each of its neighbors with the probability p (p is in the interval $[0.01, 0.1]$).

Besides forwarding the query messages, when a servent receives a query it also inspects the local site for matching services. If the local inspection results in discovering one or more services, the servent prepares a response message and sends it back towards the originator of the query. The response message traverses the path of the query message in the reverse order. To enable returning the response messages to the originator, a query originator marks its query message by a unique identifier. The servents in the path of a query cache the identifiers of the query in a short-lived buffer. When they receive a response message, they match the identifier of the response message (which is the same as the identifier of the corresponding query) against the buffered identifiers and forward the response message to the neighbor from which they have received the corresponding query.



```
POST /WSPDS.asmx HTTP/1.1
Host: micron34.usc.edu
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: http://micron34.usc.edu/SearchService
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
      <SearchService xmlns="http://micron34.usc.edu/">
            <ServiceName>Video Service</ServiceName>
            <ProviderName>VideoInfo Tech</ProviderName>
            <tModel>VideoInterface</tModel>
            <ServiceCategory>Graphics</ServiceCategory>
      </SearchService>
</soap:Body>
</soap:Envelope>
```

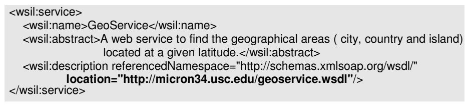
a. Query



```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
      <SearchServiceResponse xmlns="http://micron34.usc.edu/">
            <SearchServiceResult>
            http://www.videoinfotech.com/video.wsdl
            http://video.videoinfotech.com/video2.wsdl
            </SearchServiceResult>
      </SearchServiceResponse>
</soap:Body>
</soap:Envelope>
```

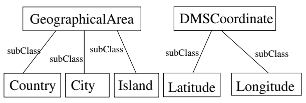
b. Response

Fig. 2. Sample Keyword-based Query/Response SOAP Messages of WSPDS Servent



```
<wsil:service>
   <wsil:name>GeoService</wsil:name>
   <wsil:abstract>A web service to find the geographical areas ( city, country and island)
                   located at a given latitude.</wsil:abstract>
   <wsil:description referencedNamespace="http://schemas.xmlsoap.org/wsdl/"
            location="http://micron34.usc.edu/geoservice.wsdl"/>
</wsil:service>
```

a. WSIL of GeoService



b. Utilized ontologies(left:geo-ont.daml, right:coord.daml)

Fig. 3. WSIL and ontologies used in GeoService

B. *Local Query Engine*

WSPDS queries allow keyword-matching queries on service name/abstarct, provider name and tModel[1](see Figure 2-a). These are the most common features currently used with discovery directories (e.g., UDDI) and service inspection documents (e.g., WSIL documents [4]) to characterize a service. This set of query features is extendible to support future interesting features (e.g. QoS) of the service.

The query engine applies the WSIL specification to inspect the local site and find services matching (string-based) with the received query. A WSIL document (e.g., Figure 3-a) lists references to the description documents (e.g., WSDL) and (possibly) UDDI records for the services available at the local site. For each service, the WSIL file also provides some metadata, such as web service name. The local query engine of the WSPDS servent parses the WSIL document of the local site and matches the query against the metadata in the WSIL document itself, as well as the metadata in the referenced service description documents and directory records. Pointers to the locations of the WSDL for the matching services are included in the response message. Due to the extensibility of WSIL specification, the query engine of the WSPDS servent can support future service description specifications.

IV. Construction of a Semantic-enabled Peer-to-Peer network (Sem-WSPDS) of WSPDS Servents

There are two major drawbacks with the primitive WSPDS network described in previous section. First, as compared to the centralized architectures, the architecture of WSPDS has higher overhead of query dissemination. With probabilistic flooding, this overhead is significantly reduced. However, as we illustrate in the following sections, we believe that a content-based peer-to-peer network, such as QDN [5], can further reduce the overhead. Second, keyword-matching is insufficient for discovering desired web services, because it ignores semantic correspondences. Since web service advertisers and requesters may look at the same service from different perspectives and express the service identity in different ways, a discovery service should rely on the semantic information to evaluate the similarity between the query and the advertised web services.

A. *Semantic-annotated Web Service Description*

There have been a number of efforts to add semantics to web service description. Ontology has been identified as the basis for semantic annotation. An ontology specifies shared expressions of concepts and agreements on the terminology/meaning for communication. DAML-S profile module [6] and semantic-annotated WSDL [7] are two emerging web service descriptions based on ontology.

DAML-S profile module is a DAML+OIL ontology for describing web services by defining “what a service does”. It can be used for discovery at the semantic level. Semantic-annotated WSDL is an XML-formatted web service description document based on WSDL, and is extended with DAML+OIL ontologies for the purpose of representing WSDL in a machine interpretable form like DAML-S profile module. Both DAML-S and semantic-annotated WSDL techniques can be utilized to add ontologies to web service descriptions and accomplish automated semantic web services discovery. Our discovery service relies on the use of semantic-annotated WSDL to describe web services interfaces, because

WSDL has been accepted as the industry standard for web service description and most of the existing web services support WSDL standards. In addition, WSDL provides communication level details of web services and numerous tools are developed based on WSDL. WSIL and semantic-annotated WSDL can provide the same capability as DAML-S profile module without adding significant complexity to the basic standards. Currently, both DAML-S and semantic-annotated WSDL only apply ontologies on the operational interfaces (i.e. input and output parameters of the operations of the web services), not on the web service names or descriptions. In this paper, we consider the semantic-matching on the operational interfaces only.

Figure 6 (see the appendix) shows the description of a GeoService web service, which finds the geographical areas, such as city, country and island, located at a given latitude. The WSDL file utilizes an approach similar to that of Sivashanmugam et al. [7] to annotate the input/output parameters of operations (e.g. *getLocByLat*) with ontology (see Figure 3-b). The input *Latitude* is restricted to the concept Latitude as defined in the coord.daml ontology, while the output is annotated with the concept GeographicalArea defined in geo-ont.daml ontology. The GeoService's WSIL file stored in the registry is shown in Figure 3-a. The service name/abstract can be queried directly from the WSIL, while input/output parameters for each operation can be retrieved by tracing the "description:location" pointer of WSIL to a semantic-annotated WSDL. A possible user query is illustrated in Figure 4-a. The query searches for service(s) that accept instances of *Latitude* as input, and generate instances of *City* as output.

B. Querical Data Network (QDN)

A QDN is a federation of a dynamic set of peer, autonomous nodes communicating through a transient-topology interconnection. An identity for each QDN node is defined based on its data content. A node joins the QDN by linking to some other QDN nodes, selecting the nodes of "similar" identity with higher probability. The nodes who know the identity of their neighbors, route the query to the neighbor that has the most similar identity to the target content (see [5] for more details about QDN). To illustrate how to build QDN connections between WSPDS servents and how to perform capability matching between the web services on the QDN, in the following sections, we consider a rather simple scenario where each node registers only one web service with one operation. Under such circumstance, the identify of each servent is defined as the ontologies associated with the input/output parameters. For the case where there are multiple web services with various operations on the same node, we map each web service operation to a virtual node and build the QDN based on the virtual nodes.

C. Communication Engine

The communication engine of a WSPDS servent exchanges SOAP-enveloped query/response messages with 1) user applications/services, or 2) other WSPDS servents. These messages are annotated with ontologies (see Figure 4 for example).

C.1 Network Setup

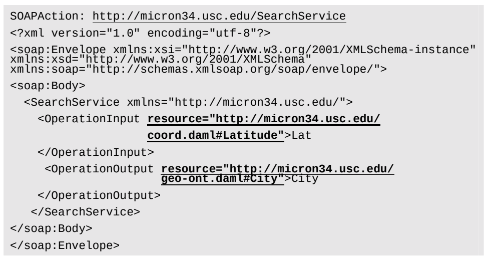
During the network setup phase, the linkages between nodes are constructed based on the data contents of the servents. A newly added node n joins the QDN by linking to some other nodes in a range geographically close to n. To select the neighbors, the new node applies a semantic matching function to evaluate the similarity between its input/output and those of the other nodes, respectively. The new node links to the nodes that have more *similar* input/output. The semantic matching function relies on the MatchMaker algorithm proposed in [8] to compute the semantic similarity. MatchMaker utilizes DAML+OIL logic to infer the similarity.

C.2 Cooperative Discovery

To discover a requested service, a SOAP-enveloped query is originated at a servent in the network (see Figure 4-a). Each servent that receives the query forwards it to the neighbor that has the most similar identity to the query (again, we use MatchMaker to calculate the similarity). In addition to forwarding the query messages, when a servent receives a query it also inspects WSIL and the semantic-annotated WSDL (whose location is specified in WSIL) on the local site for matching local services based on input and output ontologies. If the local inspection results in discovering one or more services, the servent prepares a response message and sends it back towards the originator of the query. The response message traverses the path of the query message in the reverse order.

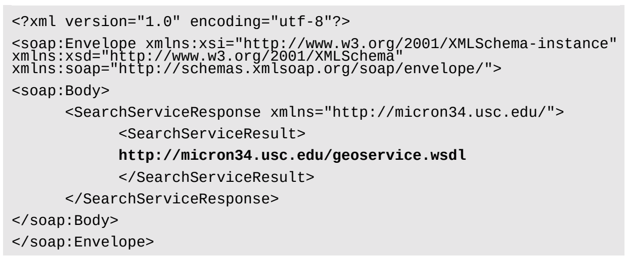
D. Local Query Engine

WSPDS queries allow semantic-matching queries on the service operational interfaces. It is extendible to support future interesting features (if semantic-enabled) such as service categories and QoS. The query engine applies the WSIL specification and the semantic-annotated WSDL to inspect the local site and find services matching with the received query. Pointers to the locations of the WSDL for the matching services are included in the response message. The match between the service



```
SOAPAction: http://micron34.usc.edu/SearchService
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
  <SearchService xmlns="http://micron34.usc.edu/">
    <OperationInput resource="http://micron34.usc.edu/
                    coord.daml#Latitude">Lat
    </OperationInput>
     <OperationOutput resource="http://micron34.usc.edu/
                      geo-ont.daml#City">City
    </OperationOutput>
   </SearchService>
</soap:Body>
</soap:Envelope>
```

a. Query



```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
      <SearchServiceResponse xmlns="http://micron34.usc.edu/">
            <SearchServiceResult>
            http://micron34.usc.edu/geoservice.wsdl
            </SearchServiceResult>
      </SearchServiceResponse>
</soap:Body>
</soap:Envelope>
```

b. Response

Fig. 4. Sample Ontology-based Query/Response SOAP Messages of WSPDS Servent

and the request is performed by comparing their input and output ontologies. The semantic-matching also uses the main idea of MatchMaker algorithm; i.e., the outputs of the query should be subsumed by the outputs of the service provided. Moreover, if inputs of the query subsume the inputs of the service, MatchMaker ranks the provided services based on their input matching. For example, consider the advertised GeoService web service shown in Figure 6 and the query shown in Figure 4-a. Their inputs match exactly, because they are restricted to the same ontological concepts (i.e. *Latitude*). Their outputs are also matching, since the query concept *City* is a subclass of the service concept *GeographicalArea*. Therefore, the web service that is able to answer the geographical areas located at a given *Latitude*, commits to provide the cities at the specified latitude.

V. Related Work

A significant amount of recent research on web services has focused on dynamic and automated web service composition [9, 10]. Towards this end, a vital step is to automatically and accurately discover the web services with desired capabilities. The idea of using peer-to-peer (P2P) and ontology to discovery web services has been proposed by [11, 12]. The P2P network utilized in our system is content-based and has a different architecture as compared to that of [11]. In addition, our approach is different from [12] both on the architecture of P2P network and the utilization of semantic enabled web service description document. Another feature that differentiate our system from theirs is that all messages exchanged among WSPDS servent are enveloped in SOAP.

VI. Conclusion

We developed WSPDS that is a decentralized discovery service with peer-to-peer architecture for the Web Services infrastructure. The primitive prototype of WSPDS is based on a variation of the Gnutella peer-to-peer network and keyword-matching between the web service descriptions. This service is currently available online. We are in the process of improving the primitive implementation based on the two concepts of content-based peer-to-peer computing and ontology-based matching. We have already developed major components (i.e., semantic-matching and QDN-linking routines) of the enhanced WSPDS, and expect to publish it for public use in near future.

References

[1] UDDI.org, "UDDI: Universal Description, Discovery and Integration of web services," 2002, http://www.uddi.org/.

[2] Gnutella, "Gnutella RFC," 2002, http://rfc-gnutella.sourceforge.net/.

[3] F. Banaei-Kashani and Cyrus Shahabi, "Criticality-based analysis and design of unstructured peer-to-peer networks as complex systems," in *Third International Workshop on Global and Peer-to-Peer Computing (GP2PC) in conjunction with CCGrid'03*, May 2003.

[4] K. Ballinger, P. Brittenham, A. Malhotra, W.A. Nagy, and S. Pharies, "Specification: Web Services Inspection Language (WS-Inspection) 1.0," November 2001, http://www.ibm.com/developerworks/library/ws-wsilspec.html.

[5] F. Banaei-Kashani and C. Shahabi, "Searchable Querical Data Networks," in *Proceedings of the International Workshop on Databases,Information Systems and Peer-to-Peer Computing in conjunction with VLDB'03*, September 2003.

[6] DAML-S Coalition, "DAML-S: Web Service Description for the Semantic Web," in *Proceedings of the First International Semantic Web Conference*, 2002.

[7] K. Sivashanmugam, K. Verma, A. Sheth, and J. Miller, "Adding Semantics to Web Services Standards," in *Proceedings of the International Conference on Web Services*, 2003.

[8] M. Paolucci, T. Kawmura, T. Payne, and K. Sycara, "Semantic Matching of Web Services Capabilities," in *Proceedings of the First International Semantic Web Conference*, 2002.

[9] J. Cardoso and A. Sheth, "Semantic e-Workflow Composition," *Journal of Intelligent Information Systems*, vol. 21, no. 3, pp. 191–225, November 2003.

[10] S. Ghandeharizadeh, C. Knoblock, C. Papadopoulos, C. Shahabi, E. Alwagait, J. L. ambite, M. Cai, C.-C. Chen, P. Pol, R. Schmidt, S. Song, S. Thakkar, and R. Zhou, "Proteus: A System for Dynamically Composing and Intelligently Executing Web Services," in *Proceedings of the International Conference on Web Services*, 2003.

[11] K. Verma, K. Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar, and J. Miller, "METEOR-S WSDI: A Scalable Infrastructure of Registries for Semantic Publication and Discovery of Web Services," *Journal of Information Technology and Management*, under review.

[12] M. Paolucci, K. P. Sycara, T. Nishimura, and N. Srinivasan, "Using DAML-S for P2P Discovery," in *Proceedings of the International Conference on Web Services*, 2003.

Appendix

```
if (message is received from user) { //message is a user query
        forward the query to the local query engine;
        forward the query to all neighbors;
}
else //message is received from a neighboring servent;
        switch (MessageDescriptor) {
                case "RESPONSE":
                        $ID$=decodeDescriptor(MessageDescriptor);
                        if ($ID$ is one of my descriptor $IDs$) {
                                merge the Result (from response) into the MergedResult with the same $ID$;
                                if (time to respond to the user query is over)
                                        return the MergedResult to user;
                        } else if ($ID$ is in my routing table)
                                forward the message according to the corresponding routing table entry;
                case "QUERY":
                        $(ID, TTL)$=decodeDescriptor(MessageDescriptor);
                        add $ID$ to the routing table;
                        send the query to the local query engine;
                        if (any matching service is found)
                                respond to the query;
                        if ($TTL > 0$)
                                forward the query to each neighbor (except the sender) with probability '$p$';
                case "PONG":
                        $ID$=decodeDescriptor(MessageDescriptor);
                        if ($ID$ is one of my descriptor $IDs$)
                                add RespondingHostAddress to the servent cache ;
                        else if ($ID$ is in my routing table)
                                route the pong message according to the corresponding routing table entry;
                case "PING":
                        $(ID, TTL)$=decodeDescriptor(MessageDescriptor);
                        add $ID$ to the routing table;
                        if (local resources are sufficient for accepting a new neighbor)
                                respond with pong;
                        if ($TTL > 0$)
                                forward the ping message to all neighbors (except the sender);
        }
```

Fig. 5. Message Processing at the Communication Engine (based on Gnutella) of a WSPDS Servent

```
<?xml version="1.0" encoding="utf-8"?>
<definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" targetNamespace="http://micron34.usc.edu/" ...>
  <types>
    <s:schema elementFormDefault="qualified" targetNamespace="http://micron34.usc.edu/">
      <s:element name="getLocByLat">
        <s:complexType> <s:sequence>
          <!-- Add ontology to input parameter using resource tag -->
          <s:element minOccurs="1" maxOccurs="1" name=" Latitude "
                resource="http://micron34.usc.edu/coord.daml#Latitude"/>
        </s:sequence> </s:complexType>
      </s:element>
      <s:element name="getLocByLatResponse">
        <s:complexType> <s:sequence>
          <!-- Add ontology to input parameter using resource tag -->
          <s:element minOccurs="0" maxOccurs="1" name=" getLocByLatResult "
               resource="http://micron34.usc.edu/geo-ont.daml#GeographicalArea "/>
        </s:sequence> </s:complexType>
      </s:element>
    </s:schema>
  </types>
  <message name="getLocByLatSoapIn">
    <part name="parameters" element="s0:getLocByLat" />
  </message>
  <message name="getLocByLatSoapOut">
    <part name="parameters" element="s0:getLocByLatResponse" />
  </message>
  <portType name="GeoServiceSoap">
    <operation name="getLocByLat">
      <input message="s0:getLocByLatSoapIn" />
      <output message="s0:getLocByLatSoapOut" />
    </operation>
  </portType>
  <binding name="GeoServiceSoap" type="s0:GeoServiceSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
    <operation name="getLocByLat">
      <soap:operation soapAction="http://microsoft.com/webservices/getLocByLat" style="document" />
      <input> <soap:body use="literal" /> </input>
      <output> <soap:body use="literal" /> </output>
    </operation>
  </binding>
  <service name="GeoService">
    <port name="GeoServiceSoap" binding="s0:GeoServiceSoap">
      <soap:address location="http://micron34.usc.edu/geoservice.asmx" />
    </port>
  </service>
</definitions>
```

Fig. 6. A WSDL document (geoservice.wsdl) annotated with ontologies(geo-ont.daml and coord.daml)