Distributed Web Service Discovery Architecture[*]

Brahmananda Sapkota Dumitru Roman Dieter Fensel
Digital Enterprise Research Institute
{firstname.lastname}@deri.org

Abstract

In this paper, we present a distributed Web service discovery architecture that is designed to be reliable, flexible and scalable. The architecture is based on the concept of distributed shared space and intelligent search among a subset of spaces. It allows the publishing of Web service descriptions as well as to submit requests to discover the Web service of user's interests. The Web service capabilities and the user requests (goal) are described using a Resource Description Framework (RDF) data model. The architecture supports integration of applications running on different resource specific devices. An application scenario is presented to illustrate the functionality of the proposed architecture.

1 Introduction

Web services offer an enabling step towards distributed computing. The Web Service Definition Language (WSDL), Universal Description, Discovery and Integration and Simple Simple Object Access Protocol (SOAP) are considered as current standards for building, building and accessing Web services. As inexpensive technical resources became available the number of Web services increased sharply. This introduced the problem of locating a particular Web service of interest from a large pool of available services. Web service may not be discovered simply because hundreds of thousands of Web services are available. Locating the required Web service, therefore, is time consuming, inaccurate, and tiresome. Thus, an automated discovery mechanism is required, that can discover the required Web service. However, the design principle of the current Web service standards undermines the automation of discovery, composition and invocation of Web services [21]. To address this problem, Semantic Web [5] technologies shall be exploited to automate the tasks of Web service discovery, composition and invocation, thus enabling interoperation between them with minimum human intervention.

In general, the purpose of service discovery is threefold: finding a service that can possibly satisfy user requirements, choosing between several services, and composing services to form a single service. It is well known that all of these tasks are carried out with the help of service descriptions well before invoking them. These service descriptions may change over time, may be distributed over locations and may be resided in heterogeneous environments. Thus, a service discovery architecture should at least support publication, finding and matching of up-to date Web service descriptions.

Current efforts in the direction of Web service discovery are focusing on Semantic augmentation of service descriptions. Some of the notable shortcomings of existing Web service technologies are synchronous communication, transient publication and no dynamic support for service discovery. It requires both service requester and service repositories (i.e. repositories of the service descriptions) to be available at the same time [6]. In an ideal situation, service requesters can not be expected to know which of the service description providers are available and where. This leads to the need for a distributed shared space where service descriptions and requests can be published, coordinated dynamically and read without knowing the origin of such descriptions. The question here is how a robust, distributed, reliable, and scalable Web service architecture can be designed such that it follows the principle of Web (i.e the REST principle). The subsequent question is how does it support resolving various heterogeneity problems transparent in Web service discovery paradigm.

In this context, this paper presents a distributed Web service discovery architecture that is designed to be robust, reliable and efficient. It is based on the concept of Shared Space [7] where applications can read and write information. Thus, it supports both synchronous and asynchronous communications, provides persistent publication of information and is scalable. It provides basic primitives required

[*]This material is based upon works supported by the EU funding under the projects DIP (FP6 - 507483), ASG (FP6 004617) and by the Science Foundation Ireland under Grant No. SFI/02/CE1/I131. This paper reflects the author's views and the Community is not liable for any use that may be made of the information contained therein.

for publishing and discovering Web service descriptions. The architecture is adaptable to the need of a particular application domain. Web service capabilities and requester's goal are described using RDF [15]. The reason behind this choice is to leverage its semantic richness that is needed for supporting dynamic automation of Web service discovery. However, the architecture can be adapted to any Ontology based languages through the use of format adapters.

This paper is further structured as follows. Different design choices of Web service discovery are presented in Section 2. Section 3 highlights the concept of shared space in distributed computing. The distributed Web service discovery architecture is described in Section 4 and access interfaces are defined in Section 5. In Section 6, an usage scenario is presented. Section 7 describes related works in the same direction and finally Section 8 summarizes this work and concludes the paper.

2 Web Service Discovery Models

The process of obtaining a set of services which can possibly fulfill a user request is called service discovery. Web service discovery, in fact, involves of two closely related tasks. The Web service descriptions have to be discovered and then they are matched against the user requirements. The way of carrying out these tasks differs between architectures. Different types of Web service discovery model exist subject to availability of Web service descriptions. It can be static, centralized or decentralized.

The simplest of all is the static web service discovery model. In this model, Web service descriptions are stored locally and their content remain *static*. In the centralized discovery model, Web service descriptions are published to a directory service. Requests are forwarded to the directory service and Web service descriptions are obtained. This type of model is also known as as service join lookup [2]. The service discovery model where no centralised directory service exist is called a *dynamic* service discovery model. These models use distributed technologies such as data federation [20], P2P [22], and agent based systems [17]. We adopt the distributed model based on [6] in our architecture as it provides various desired features such as scalability, reliability, no central control etc.

3 Shared Space

This section aims to introduce the notion of shared space computing and their variations that motivates the use of shared space concepts in distributed Web service discovery architecture. Shared space concepts have been used to explore applications on distributed systems [8, 16, 9, 11] which are relevant to the area of Web service discovery. The space-based systems are said to be loosely coupled since applications communicate via virtual spaces. Space-based architecture offers a number of desired features such as robustness, scalablility, persistency and adaptability. Each Web service and space may exist on its own anywhere on the Internet. Figure 1 represents an abstract shared space architecture where processes share messages by writing and reading.

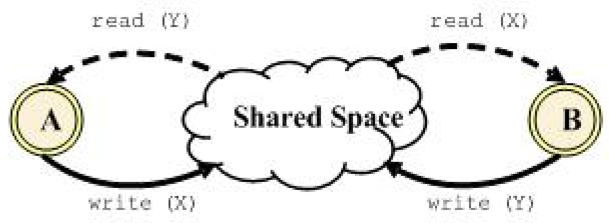

Figure 1. Shared space communication

A well explored shared space concept is publish-subscribe interaction mechanism whose functionality is centered around the concept of an event [8]. Information is communicated by publishing an even or topic and subscribing for a particular event. All published events are propagated to its subscribers. It has been widely recognised as a promising communication infrastructure in distributed environment [13].

The tuple space concept was introduced in Linda [10]. In tuple space, information is communicated by writing and reading an ordered set of typed fields called tuples. When tuples are no longer needed they can be deleted. Since it decouples communication with respect to time, location and space it has been adopted in different systems [1, 24]. The tuple space operates on simple data model and thus do not support Semantic augmentation of information. The compelling three dimensional decoupling feature, however, attracted the interest of research communities to opt for semantic tuple spaces [9, 6, 14].

4 Architecture

The distributed Web service discovery architecture is presented starting with the design concepts followed by component descriptions. In Section 4.1, our design principle and assumptions are presented, Section 4.2 describes minimum desired components that builds up the discovery architecture, and finally Section 4.3 presents system functionality.

4.1 Design Concept

The distributed Web service discovery architecture presented in this paper is based on the concept of persistent publication of data in general and tuple based shared space in particular. The choice is made as it is promises different design advantages i.e. it decouples communication in three prevailing dimensions: time, location, and access. Adopting the shared space concept, the basic required components of the distributed Web service discovery architecture comprise Discovery Manager, Query Parser, Space Reader, and Writer. Figure 2 represents a high level distributed Web service discovery architecture. The tag attached with each participants in the network denotes their Web service description published on the shared-space.

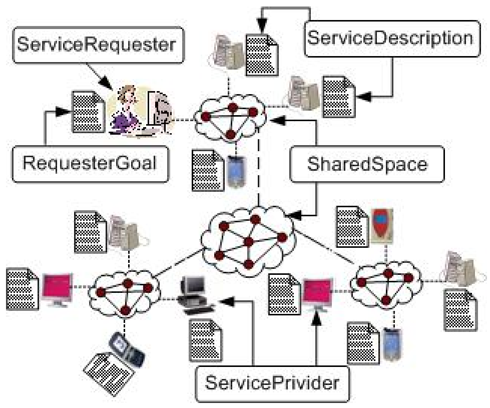

Figure 2. A space-based system overview

This architecture allow users to create a new virtual shared space, or use an existing one if available. Messages can be written to or read from the space. Any user (either a service provider or a requester) can have access to the information available on the space (subject to security policy). Thus, when looking for a Web service, a request can be sent to the virtual shared space instead of sending to individual Web service description repositories. The added advantage of this approach is that it reduces the communication overhead and the result obtained are more precise. The detailed architecture and its components are described in next section.

4.2 Component Descriptions

Based on the above mentioned architecture concepts the architecture components and their interactions are introduced in this section. Figure 2 depicts the proposed architecture. These components are loosely coupled and are pluggable. This allows replacement of existing implementations over time with alternative or more expressive implementations as well as new components.

Discovery Manager: Discovery manager is a gateway to distributed Web service discovery and provides access interfaces serving as a point of interaction. It receives requests from the user and returns response to the user. When

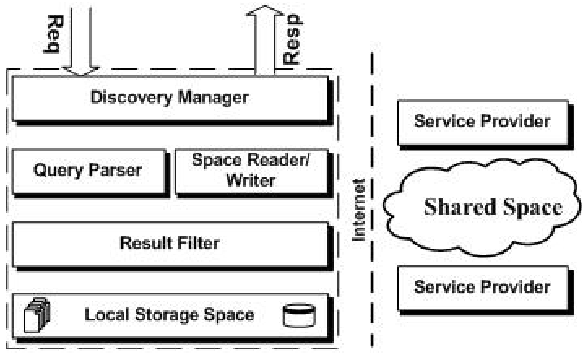

Figure 3. Discovery architecture

a request is received, it schedules a job for query parser.

Query Parser: Query parser is responsible for parsing each incoming requests. It determines the requests type (e.g., read, write, take) and forwards them to space reader or space writer.

Space Reader/Writer: The job of finding the available spaces, writing messages to and reading messages from a space is handled by space reader/writer component. It consist of two sub-components, reader and writer. Reader finds and reads messages from the shared space and writer writes messages to the spaces. It is the job of the writer to keep status of requests to the local storage space.

Matchmaker: The concrete matchmaking between the requester goal and available Web service descriptions is done by the matchmaker component. It receives the set of Triples from the space reader, obtains goal descriptions from the local storage space and performs matchmaking between them.

Local Storage Space: The local storage space is used to store the intermediary data produced by the distributed Web service discovery system. It is also used for storing interface descriptions of the internal components and information about shared spaces.

4.3 System Functionality

The functionalities supported by this architecture are publishing Web service descriptions, reading them from the shared space and discovering the required Web service. The discovery manager accepts Web service descriptions and presents it the space writer for publishing it on the shared space. The space writer writes this service description to the shared space. The descriptions of the published Web services are obtained by querying the shared space.

It is important to note that a Web service may not always be able to fulfill a requester's goal. In a practical situation, a requester's goal normally requires a multitude of services and a service may satisfy only a part of the requester's goal. Thus, for satisfying a requester's goal, multiple services may have to be discovered. For this reason each read requests has to be forwarded to the query parser for parsing and identifying related sub-goals.

The query parser generates key concepts from the original goal. For each key concepts, a sub goal is made and presented to the space reader according to goal decomposition algorithm presented in [21]. The space reader then reads Web service descriptions matching these sub-goals from the shared space. These descriptions are merged and presented to the service requester.

5 Interfaces

For the distributed Web service discovery architecture to work properly interfaces has to be defined and implemented. In the following the interfaces of main components shown in figure 3 are described and their implementation approach is discussed. The interfaces presented here are the basic required interfaces, and they can be extended to support more advanced operations without hindering the core idea of the architecture.

RequestEntry – *RequestEntry* is the public interface exposed by discovery manager to its users. All communication with discovery manager takes place through this interface. It implements the common web interface, (i.e. HTTP_GET, HTTP_PUT, HTTP_POST and HTTP_DELETE) and supports *publish*, *retrieve*, *create* and *destroy* operations. The semantics of these operations are the following the following.

Method Summary	
SpaceID	*create* ( )
void	*create* (SpaceID)
void	*destroy* (SpaceID)
void	*publish* (RDF, SpaceID)
void	*publish* (RDF)
void	*retrieve* (RDF, SpaceID)
RDF	*retrieve* (RDF)

Parse – *Parse* is the interface provided by query parser that is used to invoke query parser by the discovery manager component. It supports the *parse* operation that generates a collection of RDF Triple(s) from a given RDF graph and vice versa. Following is the semantics of this operation.

Method Summary	
Triple [ ]	*parse* (RDF)
RDF	*parse* (Triples [ ])

Reader – The reader component provides a *read* operation made accessible through its *Reader* interface. In the following the semantics of this operation is shown.

Method Summary	
Triple[ ]	*read* (Triple)
Triple[ ]	*read* (Triple, SpaceID)

Writer – A *write* opration is provided by the writer component and it is made accessible through the *Write* interface. The semantics of this operation is the following.

Method Summary	
void	*write* (Triple[ ], SpaceID)
Triple[ ]	*write* (Triple[ ])

Filter – The request filter component supports *filter* operation made available through its *Filter* interface. Given a set of Triples, filter operation filter outs the triples not relevant to the goal and returns only relevant triples. The semantics of this operation is as follows.

Method Summary	
Triple[ ]	*filter* (Triple[ ], Triple [ ])

Storage – The local storage component supports *save*, *retrieve*, *delete* and *update* operations. These operations can be accessed via *Storage* interface provided by the local storage space component. Following is the semantics of these operations.

Method Summary	
boolean	*Save* (Triple[ ])
Triple[ ]	*retrieve* (Triple)
boolean	*delete* (Triple[ ])
boolean	*update* (Triple[ ])

6 Application Scenario

The interfaces mentioned before are used in this section for illustrating the functionality and usage scenario of the proposed architecture. Based on the WSMO (Web Service Modeling Ontology) [19] model, Web service and requester's goal are described in RDF.

Publishing a Web service description – MoParts is a motor part manufacturer. It sells motor parts to different motor manufacturers. It has traditional customers' but is willing to expand the market share. Their capabilities are described in RDF as shown in Figure 4. To publish this service description, MoParts invokes *create* (MoParts) operation through RequestEntry interface and creates a virtual shared space with *SpaceID* MoParts. After creating this virtual shared space, it invokes *publish* (RDF) operation and publishes its service description. When the discovery manager receives a publish request, it requests the query parser to generate a collection of RDF Triples from the given RDF graph. After generating a collection of RDF Triples, the *write* (Triple[ ], MoParts) operation of the writer component is used to

```
<?xml version=‘‘1.0’’?>
<rdf:RDF xmlns:rdf=‘‘http :// www.w3.org/1999/02/22−rdf−syntax−ns#’’
         xmlns:dc=‘‘ http :// purl .org/dc/elements /1.1/’’
         xmlns:dt=‘‘ http  ::// www.wsmo.org/ontologies/dateTime#’’
         xmlns:tc=‘‘ http  ::// www.wsmo.org/ontologies/trainConnection#’’
         xmlns:po=‘‘ http  ::// www.wsmo.org/ontologies/purchase#’’
         xmlns:loc=‘‘ http  ::// www.wsmo.org/ontologies/location#’’
         xmlns:ws=‘‘ http :// www.example.org/resources/ws.wsml’’
         xmlns:mediator=‘‘ http :// www.example.org/mediators#’’>

<rdf:Description  rdf:about=‘‘ http :// www.example.org/PartsSalesService’’>
   <ws:hasNFP rdf:nodeID=‘‘nfp’’/>
   <ws:capability>
      <ws:Description rdf:nodeID=‘‘precondition’’/>
   </ws:capability>
   <ws:capability>
      <ws:Description rdf:nodeID=‘‘postcondition’’/>
   </ws:capability>
   <ws:useMediator rdf:nodeID=‘‘usedMediators’’/>
   <ws:useInterface  rdf:nodeID=‘‘ usedInterface ’’/>
</rdf:Description>
<rdf:Description  rdf:nodeID=‘‘nfp’’>
   <dc:title> Motor Parts  Sales  Service  </dc:title>
   <dc:publisher> Motor Parts  Trading  Company </dc:publisher>
   <dc:description> This service   sells  motor  parts  </dc:description>
   <ws:quality> High </ws:quality>
</rdf:Description>
<rdf:Description  rdf:nodeID=‘‘usedMediators’’>
   <ws:usesMediator> mediator:ProcessMediator.wsml </ws:usesMediator>
   <ws:usesMediator> mediator:ConceptMediator.wsml </ws:usesMediator>
   <ws:usesMediator> mediator:CountryMediator.wsml </ws:usesMediator>
   <ws:usesMediator> mediator:CurrencyMediator.wsml </ws:usesMediator>
   <ws:usesMediator> mediator:TransportMediator.wsml </ws:usesMediator>
</rdf:Description>
<rdf:Description  rdf:nodeID=‘‘precondition’’>
   <ws:hasAxiom> PartsList </ws:hasAxiom>
   <ws:hasAxiom> DeliveryLocation </ws:hasAxiom>
   <ws:hasAxiom> RequriedDeliveryDate </ws:hasAxiom>
   <ws:hasAxiom> OfferedPrice </ws:hasAxiom>
</rdf:Description>
<rdf:Description  rdf:nodeID=‘‘postcondition’’>
   <ws:hasAxiom> AvailablePartsList </ws:hasAxiom>
   <ws:hasAxiom> NeededDeliveryDays </ws:hasAxiom>
   <ws:hasAxiom> Price </ws:hasAxiom>
</rdf:Description>
<rdf:Description  rdf:nodeID=‘‘ usedInterface ’’>
   <ws:interface>  PartsSaleseInterface  </ws:interface>
   <ws:choreography> PartsSalesChoreography </ws:choreography>
</rdf:Description>
</rdf:RDF>
```

Figure 4. Web service description in RDF

publish the request onto a virtual shared space identified by MoParts.

Achieving a goal – SeMa is a motor trading company. It assembles parts for building custom cars and sells them at a competitive price. The company never keeps stock of motor parts but buys them when it decides to build a custom car. It makes a list of necessary parts and describes them as in Figure 5. Since it does not know where the required parts are available, it invokes *retrieve* (RDF) operation of through RequestEntry interface in an attempt to achieve the goal. This goal is parsed by the query parser and a set of RDF Triples are generated. This set of RDF Triples is then passed to the reader component via *read* (Triples [ ]) operation and a set of result Triples is obtained. The result set goes through the filter that filters out irrelevant triples.

```
<?xml version=‘‘1.0’’?>
<rdf:RDF xmlns:rdf=‘‘http :// www.w3.org/1999/02/22−rdf−syntax−ns#’’
         xmlns:dc=‘‘ http :// purl .org/dc/elements /1.1/’’
         xmlns:loc=‘‘ http  ::// www.wsmo.org/ontologies/location#’’
         xmlns:g=‘‘ http :// www.example.org/resources#’’
         xmlns:mediator=‘‘ http :// www.example.org/mediators#’’>

<rdf:Description  rdf:about=‘‘wsml:partsPurchaseGoal.wsml’’>
   <g:hasNFP rdf:nodeID=‘‘nfp’’/>
   <g:goal rdf:nodeID=‘‘postcondition’’/>
   <g:importsOntology rdf:nodeID=‘‘importedOntology’’/>
</rdf:Description>
<rdf:Description  rdf:nodeID=‘‘nfp’’>
   <dc:title>Motor Parts Purchase  Goal</dc:title>
   <dc:publisher>SeMo Trading Company</dc:publisher>
   <dc:description>Parts purchase  goal</dc:description>
   <g:quality>High</g:quality>
</rdf:Description>
<rdf:Description  rdf:nodeID=‘‘importedOntology’’>
   <g:importsOntology>mediator:ProcessMediator.wsml</g:importsOntology>
   <g:importsOntology>mediator:ConceptMediator.wsml</g:importsOntology>
   <g:importsOntology>mediator:CountryMediator.wsml</g:importsOntology>
   <g:importsOntology>mediator:CurrencyMediator.wsml</g:importsOntology>
   <g:importsOntology>mediator:TransportMediator.wsml</g:importsOntology>
</rdf:Description>
<rdf:Description  rdf:nodeID=‘‘postcondition’’>
   <g:hasAxiom>purchasingParts</g:hasAxiom>
   <g: partsDefinition  rdf:nodeID=‘‘ definition ’’/>
</rdf:Description>
<rdf:Description  rdf:nodeID=‘‘ definition ’’>
   <dc:name>Wheel</dc:name>
   <g:quantity>100</g:quantity>
   <g:deliveryDays>7</g:deliveryDays>
   <g:pricePerUnit  rdf:nodeID=‘‘ unitPrice ’’/>
</rdf:Description>
<rdf:Description  rdf:nodeID=‘‘ unitPrice ’’>
   <g:maxPrice>99.99</g:maxPrice>
   <g:currency>Euro</g:currency>
</rdf:Description>
</rdf:RDF>
```

Figure 5. Goal description in RDF

The final set of triples thus obtained goes through the query parser and a RDF graph is obtained which is then presented to the requester.

7 Related Works

The problems pertaining to Web service discovery have long been taking attention of both academia and industry. In [4], a Peer-to-Peer Web service approach have been proposed. It annotates WSDL files with semantic description based on WSDL-S framework making WSDL-S agnostic to other ontology based languages. One of the problems of this approach is that it adds additional overhead of annotating WSDL descriptions. The use of single gateway peer leads to the problem of a single point of failure as well as communication bottleneck. A scalable discovery mechanism has been presented in [23], [18]. However, they are unable to include resource limited devices into the Web service discovery picture. The architecture proposed by us works on the semantic level and allows interlinking of resources through the use of RDF data model.

Jini [2] provides a discovery framework based on

JavaSpace [1] and Java RMI (Remote Method Invocation). Services register with lookup servers and they can be discovered by querying the lookup server. Jini search mechanism uses the Java serialized object matching thus leading to false negative problems. On the other hand, our architecture uses pure Web standard and provides persistent storage of messages. There is no need of object serialization and thus no false negative can occur.

Neuron [12] shares common idea with our architecture. It provides a shared object space on top of a Peer-to-Peer network. When a service is advertised a virtual shared space is created. The advertised descriptions are securely encrypted. This framework, however, does not provide semantic support. Service requester have to obtain security key from its provider to view the service description which is a undesirable feature of the distributed system. In addition it adds extra communication overhead. The architectural solution we provides full semantic support. The services can be queried by providing a RDF description. In addition, the service composition is easier as we retrieve all relevant triples from the shared space.

8 Conclusions and Future Direction

In this paper we presented a discovery architecture that is designed to be distributed, scalable, reliable and address heterogeneity problems stemming from resource limited devices such as PDA, mobile phone etc. The interfaces the functionalities of the proposed architecture are described. The use of shared space makes this architecture reliable, fault tolerant, scalable as each node in the Internet can be imagined hosting a shared space. It enables easy composition of Web service as the requests are handled at RDF Triple level.

In the current implementation, shared space is centralized. It allows to create virtual spaces and query those virtual spaces. As part of our next step, we intend to upgrade the current implementation from centralized shared space to distributed shared space. While doing so, WSMX (Web Service Execution Environment) [3] will be used as a testbed platform.

References

[1] Sun Microsystems, Inc. *JavaSpace Specification, Revision 0.4*, 1997.

[2] Sun Microsystems, Inc. *Jini Architectural Overview, Technical Report,* 1999. http://www.sun.com/software/jini/whitepapers/architecture.pdf.

[3] Web Service Execution Environment. 2005. http://www.wsmx.org.

[4] F. Banaei-Kashani, C.-C. Chen, and C. Shahabi. WSDP: Web Services Peer-to-Peer Discovery Service. *International Conference on Internet Computing*, pages 733–743, 2004.

[5] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, May 2001.

[6] C. Bussler. A Minimal Triple Space Architecture. *In Proceedings of the WSMO Implementation Workshop*, June 2005.

[7] N. Carriero and D. Gelernter. A Computational Model of Everything. *CACM*, 44(11):77–81, 2001.

[8] P. T. Eugster, P. A. Guerraoui, and A. M. Kermarrec. The Many Faces of Publish/Subscribe. *ACM Computing Surveys*, 2003.

[9] D. Fensel. Triple Space Computing. *Technical Report, Digital Enterprise Rresearch Institute (DERI)*, 2004.

[10] D. Gelernter. Generative Communication in Linda. *In Proceedings of ACM Transactions on Programming Languages and Systems*, 7(1):80–112, January 1985.

[11] D. Gelernter and N. Carriero. Coordination Languages and Their Significance. *ACM Computing*, 1992.

[12] H.-C. Hsiao and C.-T. King. Neuron - A Wide-Area Service Discovery Infrastructure. *In Proceedings of the Interntaltional Conference on Parallel Processing (ICPP'02)*, 2002.

[13] Y. Huang and H. Garcia-Molina. Publish/Subscribe in a Mobile Environment. *In Proceedings of the MobiDE*, 2001.

[14] D. Khushraj, O. Lassila, and T. W. Finin. sTuples: Semantic Tuple Spaces. *In Proceeding of MobiQuitous*, 2004.

[15] G. Klyne and J. Carroll, editors. *Resource Description Framework (RDF): Concepts and Abstract Syntax*. W3C Recommendation, February 2004. http://www.w3.org/TR/rdf-concepts/.

[16] F. Leymann. A Practioners Approach to Database Federation. *In Proceedings of 4th Workshop on Federated Databases*, 1999.

[17] M. Montebello and C. C. Abela. DAML Enabled Web Services and Agents in the Semantic Web. *Revised Papers from the NODe 2002 Web and Database-Related Workshops on Web, Web-Services and Database Systems*.

[18] M. Pistore, F. Barbon, P. Bertoli, D. Shaparau, and P. Traverso. Planning and Monitoring Web Service Composition. *In Proceedings of the Workshop on Planning and Scheduling for Web and Grid Services*, June 2004.

[19] D. Roman, H. Lausen, and U.Keller, editors. *Web Service Modelling Ontoloty (WSMO)*. WSMO Deliverable, v1.2, 2005.

[20] P. Rompothong and T. Senivongse. A Query Federation of UDDI Registries. *ISICT*, 2003.

[21] B. Sapkota, L. Vasiliu, I. Toma, D. Roman, and C. Bussler. Peer-to-Peer Technology Usage in Web Service Discovery and Matchmaking. *In Proceedings of the 6th International Conference on Web Infromation Systems Engineering (WISE 05) (to appear)*, November 2005.

[22] M. Schlosser, M. Sintek, S. Decker, and W. Nejdl. A Scalable and Ontology-based P2P Infrastructure for Semantic Web Services. *In Proceedings of the Second International Conference on P2P (P2P'02)*, 2002.

[23] I. Toma, B. Sapkota, J. Secuila, J. M. Gomez, D. Roman, and C. Bussler. P2P Discovery Mechanisms for Web Service Execution Environment. *Second WSMO Implementation Workshop*, 2005.

[24] P. Wyckoff, S. McLaughry, T. Lehman, and D. Ford. T Spaces. *IBM Systems Journal*, 37(3):454474, 1998.