Scalable Architecture for Web Service Discovery

Brahmananda Sapkota, Sanaullah Nazir, Manfred Hauswirth
DERI Galway, NUI Galway
Galway, Ireland
firstname.lastname@deri.org

Tomas Vitvar
STI Innsbruck
University of Innsbruck
Innsbruck, Austria
firstname.lastname@sti2.at

ABSTRACT

Web services offer an enabling step towards enterprise integration over the Internet. Service discovery is the fundamental task of finding and assembling services to meet a request. In this paper a scalable service discovery architecture is proposed. The scalability is mainly achieved by employing the shared semantic space for discovery and coordination of resources distributed over the shared spaces. The shared spaces are virtual spaces which can be hosted at any node or server on the Internet. The system self-organizes into virtual groups that empower the capability of discovery mechanism.

Keywords

Semantic Web, Semantic Web Service, Scalable Web Service Discovery, Semantic Shared Space, Tuple Space

1. INTRODUCTION

Web services offer an enabling step towards distributed computing. Web Service Definition Language (WSDL), Universal Description, Discovery and Integration (UDDI) and Simple Object Access Protocol (SOAP) are the building blocks of current Web services. Web services are built and deployed independently. UDDI is used for indexing service descriptions encoded in WSDL. One can manually obtain required service description from a UDDI registry and invoke them. However, the result of such discovery is only the specifications of the Web services but not necessarily the one user is looking for. The state of any particular Web service is unknown. The increasing number of Web services also indicates that service descriptions will not remain in a single repository nor can be indexed using a single registry. These systems will have difficulty in supporting scalability, automating and discovering services from heterogeneous environments. Thus, we realize that a mechanism is required for efficiently locating and utilizing the required Web service. Through the use of ontologies, Semantic Web technology makes service descriptions machine-readable. Providing a shared memory space, Tuplespaces support persistent publication and strong decoupling between applications in terms of reference, time and space, ensuring scalable communication. A combination of them could potentially improve service discovery.

The UDDI based approaches, for example, for publishing Web service description greatly restrict the query recall and thus affects the precision of the results. The dynamism envisioned in Semantic Web cannot be achieved in UDDI as it restricts the use of data in a closed environment. To cover a larger user base and to encourage the potential use of Semantic Web services, it is vital to incorporate open and close environments in a discovery system. This leads to the need of a mechanism which can ensure the discovery of appropriate services, if available. In order to ensure this, Web service descriptions need to be discovered from repositories distributed over the network. Thus, the discovery architecture should incorporate the functionality of collecting, distributing, and disseminating Web service descriptions in the network.

This document describes a distributed Semantic service discovery mechanism that is designed to be highly distributed, robust, reliable and scalable. The primary focus of this paper is on service discovery coupled with Semantic Web [1] and shared space [2] principles. We propose to create virtual groups of related Web services in order to accelerate the query process and to improve the quality of results. This implies that the descriptions of similar Web services are kept in one virtual group. The similarities between two Web services can be evaluated based on the services they provide and their usage constraints. This is achieved by employing a self-organization algorithm which ensures that a service description is published to the right virtual group. The virtual groups are maintained and made available through the Semantic shared spaces. The use of shared space, in addition, allows persistent publication and execution of service descriptions. Further, it decouples the service providers from their users enabling them to focus on their own business logic.

1.1 Application Scenario

To depict the applicability of our architecture for web service discovery, we present a scenario which has services providers belonging to various domains e.g transportation, postal and health etc. Users interested in services which provide flight reservation would be interested in getting quick response from service providers providing relevant services. A user would interact with the broker and send his request to it. The broker will look for the services by sending re-

quest to service providers which belong to the domain of interest of the user request. Figure 1, portrays the application scenario. It shows service providers belonging to different domains are registering and publishing their services with the broker. The users interact with the broker in order to retrieve services from the service provider. The broker matches users requests with the published services of the service providers.

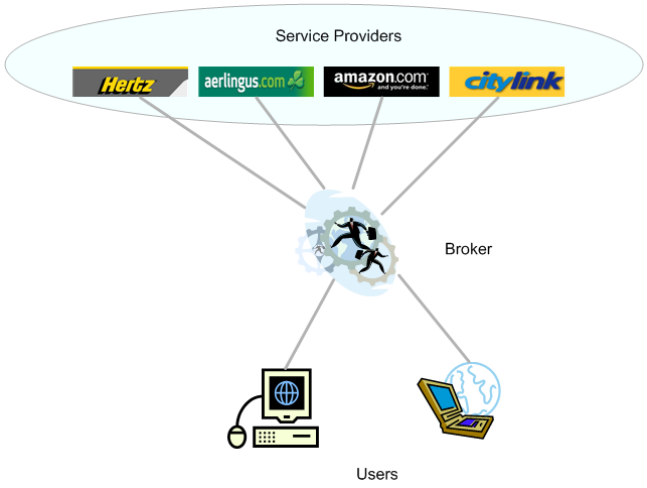


Figure 1: Application Scenario

In Section 2, Semantic Web services and shared-space computing paradigm are introduced. The usual Web service discovery approaches are described in Section 3. The Semantic shared space is described in 4. Section 5 takes this further with a proposal for distributed discovery architecture and Section 6 presents a distribution algorithm employed in this architecture. The usability of the proposed architecture and algorithm are illustrated with the help of a running example in Section 7. The proposed architecture is evaluated and results are presented in Section 8. Relevant existing works are discussed in Section 9 and this paper is concluded in Section 10 highlighting the direction for future works.

2. BACKGROUND

The background technologies such as Semantic Web services, shared tuple space, used in this paper are briefly described in the following subsections.

2.1 Semantic Web

Web services [3] are proposing a uniform and standardized way to access services over the Web. They address stringent business problems like integration and automation of business processes. Their fundamentals rely on standards like SOAP, WSDL and UDDI. However, all of these standards operate at a syntactic level and, therefore, a scalable solution for business problems cannot be achieved without human support. Realizing such a solution will require richer descriptions that can be processed by machine in a meaningful way, thus enabling the automation of service related tasks such as service discovery, composition and execution.

With current research around Semantic Web [1], Semantic Web services aim at describing the various aspects of a Web service using explicit, machine-understandable semantics, enabling the automatic location, combination and use of Web services [4, 5]. The main principle of Semantic Web services [6–8] rests on publishing service ontologies, dynamic discovery of those ontologies, mediation between them, when needed, to respond to a requester's goal. A web service is defined in term of its capability and conditions. A dynamic discovery involves automatically and dynamically locating Web services that can fulfill user requirements. If the user requirements cannot be fulfilled by a single Web service, multiple Web services may need to be used to fulfill the requirements. Mediation involves mediating between the Web services or between a Web service and user requirements, if there is some difference on the protocol or data level.

2.2 Tuple Space

The shared space concepts have been used to explore applications on distributed systems [2, 9–11] The shared-space based systems are loosely coupled because applications communicate via shared virtual spaces. The features offered by shared-space architecture are robustness, scalability, persistence and adaptability. Each Web service and space may exist on its own anywhere on the Internet.

A well explored shared-space concept is publish-subscribe interaction mechanism whose functionality is centered around the concept of an event [10]. Information is communicated by publishing an event or topic and subscribing for a particular event. All published events are propagated to their subscribers. It has been widely recognized as a promising communication infrastructure in distributed environment [9].

The concept of shared-space was first introduced in Linda [2] as tuple space. 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 [12, 13]. A tuple space, however, operates on simple data model and thus does not support semantic augmentation of information. The compelling three dimensional decoupling features, however, attracted the interest of research communities and motivated them to opt for semantic tuple spaces namely triple space [9]. The visible advantages of using this in Web service paradigm are threefold. Firstly, the Web service paradigm advances to the Web paradigm, i.e., the persistent publication of information. Secondly, the communication is asynchronous [14]. Thirdly, shared space provides middleware support hiding from internal application complexities.

3. WEB SERVICE DISCOVERY

Web service discovery is the process of locating web services which can possibly fulfill the user requirements. Traditionally the protocol for discovery in syntactic world is UDDI which is primarily based on keyword search on a global Web service registry. There are many disadvantages of the syntactic or keyword-based discovery approach. It lacks dynamic discovery i.e. when a single service is not able to fulfill a goal, it is not possible to predict, without human intervention, if the goal can be fulfilled. The search on a global service registry may introduce scalability issues.

In semantic world, approaches like WSMO [6], OWL-S [7], METEOR-S [8] etc. have tried to address this problem by describing the services in a way that makes them machine readable. Efforts include development of expressive languages based on artificial intelligence technology including RDF [15], RDF-S [16] and most recently OWL [17] and

WSML [18]. Similarly complex reasoners have been developed to automate the process of web service discovery and composition.

In this approach, service provider creates a WSDL describing the service it is offering. This WSDL is published to some registry (e.g. UDDI or ebXML). The requester of the service finds this WSDL by browsing the registry. Having found this description, the requester then invokes the service using SOAP protocol.

In tuple spaces, the description of an offered services is described in the form of tuple containing an ordered set of elements. These elements describe capability, interfaces, and conditions for invoking this service. The requester needs to formulate a template describing the needed functionality. This template is sent to tuple space and matching service is discovered.

4. SEMANTIC SHARED SPACE

The Semantic shared-space is created in order to coordinate the communication between distributed applications. This space balances the load between participating applications. In the following subsections the Semantic shared-space is introduced and its relation with Web services is illustrated.

4.1 Semantic Shared Space

The Semantic shared space is a virtual space which is created through the network of distributed shared spaces. These shared spaces are uniquely identified by URL and can be hosted in any node or server in the Internet. These shared-spaces are connected with each other in such a way that the nodes having similar information form a group (SemSet). Two services are considered to have similar interest if they share similar interfaces, domain ontology and input-output relations. Actual algorithm for defining these similarities is an implementation issue and is not discussed in this section.

The basic principles behind this grouping is to partition the global space into sub spaces of related information but it is also possible to have overlapping interests. This means that a node can take part in different interest groups. This overlap between interests is utilized for creating a virtual coordination space (CoSpace) which is used for coordination between different interests groups. Therefore, every node is reachable from any node in the network. The coordination space is also utilized for active replication of data which is required to prevent unpredictable losses from system failures. This means that the Semantic shared space is failure resilient. It also keeps history of all flow of messages and requests to and from the shared spaces. The grouping principle is further discussed in Section 6.1.

The internal data model used in this network is *named graph* [19]. The Semantic shared space allows persistent publication of data in general and tuple based shared space in particular. The promise of such shared space is to decouple applications in three prevailing dimensions: time, location, and access reference. Three main system parts exist in a Semantic shared space. One major system part is user application which can be a reader or writer. Another system part is the space provider which can host a number of shared spaces and the third system part is the protocol used for communication. The protocol used for communication between shared spaces is based on HTTP standard.

The Semantic shared space mentioned here is similar to that in [20]. However, the architecture presented in [20] is minimal i.e. a client has to connect to many different shared space servers. We extend the present minimal triple space architecture to accommodate interaction between the triple space servers possible by connecting them in a hybrid Peer-to-Peer network (P2P). In the P2P architecture two hosts can communicate with each other through the same HTTP protocol. According to this protocol the possible syntax for reading and writing operations are:

```
1 http://URL/namedGraph
2 http://URL/URI
```

Listing 1: Semantic shared space basic operations

where URI is a unique identifier of a *named graph* $\mathcal{N}_g$. The first operation is the write operation where the URL of $\mathcal{N}_g$ to be written is provided followed by the named graph itself. The second operation is the read operation that provides the URL of the Semantic shared space and the URI of the named graph to be read.

4.2 Semantic Service Space

The location for publishing and accessing data is known as Semantic shared space which is based on the philosophy that a Web service can be published and read independently. Named graphs are written and read with this URL as a storage location. The query language used for reading named graphs is based on SPARQL protocol [21].

The notable advantages of Semantic shared space as mentioned before has major role to play in the (Semantic) Web services (SWS) world such as a global semantic repository, persistent and semantic communication infrastructure. Enabling asynchronous communication and using semantically enriched resources, much of the process can be automated achieving scalability significantly and managing communication complexity. That is, the writing or reading applications will always be notified in the event of unsuccessful execution of the requests. Thus, information will never be lost in case of system failure. In addition, the history of operations over the shared space can be used for reusing already available Web services. Due to its internal data model, it is easy to maintain provenance information. It can ensure that the information arrived from the said source cannot be denied. Finally, the shared space plays a middleware role for discovering required services in SWS paradigm.

5. DISCOVERY ARCHITECTURE

The distributed architecture presented in this paper consists of loosely coupled components and is based on SOA design principles. The architecture of the Semantic shared space is not included in detail due to brevity. Therefore, the focus here is on the discovery architecture as shown in Figure 2. The basic required components of this architecture are: Access Interface, Communication Manager, Parser, Result Aggregator, Monitor, Space Invoker, Result Filter and Load Distributor. It provides a HTTP based access interface.

This architecture allows its 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 local security policy). Thus, when looking for a Web service, a request is

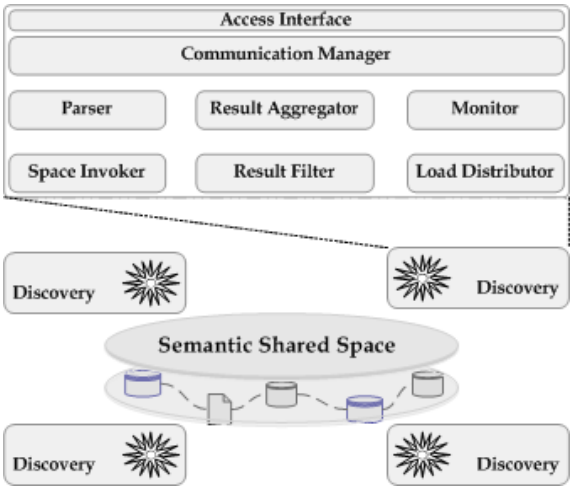


Figure 2: Distributed Discovery architecture

sent to the virtual shared space instead of sending to individual Web service description repositories. The components of this architecture are described in next section.

5.1 Component Description

The system 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. The functionalities of these components are described below.

Communication Manager

The *Communication Manager* separates the scope of access interface from that of internal functionalities of the system. Its major responsibility is to deal with sending and receiving requests and results to and from the discovery system. Its external behaviour is accessed through the *Write* and *Read* interfaces. The requests are encoded using a goal template $\mathcal{T}_G$, where a goal represents both the *write* and the *read* operations.

Parser

The main functionality of a *Parser* is to obtain the goal template $\mathcal{T}_G$ from the communication manager via the load distributor, parse it and store the obtained data in the local repository for future use. A $\mathcal{T}_G$ can consist of a simple or a complex goal. The parser needs to decompose these complex goals because of various reasons such as: the information required to evaluate the $\mathcal{T}_G$ is distributed; $\mathcal{T}_G$ consists of multiple sub-goals; it requires mutually exclusive graph composition; and the $\mathcal{T}_G$ can be executed only partially. Thus based on the named graphs $\cup\mathcal{N}_g$ defining $\mathcal{T}_G$, it is decomposed into sub-goals $\{\mathcal{T}_g\}$ such that $\cup_i^n \mathcal{T}_g^i \equiv \mathcal{T}_\mathcal{G}$ as described in [22]. Following simple algorithm is used for such decomposition:

```
1  {T_g} = NULL
2  for each N_g of T_G
3      T_g ∪ extract N_g
4  return {T_g}
```

Listing 2: Sub goal template generation Algorithm

Result Aggregator

This component is responsible for aggregating the result subgraphs received from *Space Invoker* and to complete a result template or a result graphs. The various possible complete result graphs are sent to the *Filter* component for further actions.

Monitor

Observes and oversees the overall resource consumed locally while executing requests. A monitor periodically checks the status of discovery servers by probing them with a particular kind of message and expecting a reply from them not later than a configured response time. Based on the response, it takes appropriate action. If a response does not come the server is marked as **DEAD** and the next probe is sent after the configured down time. The response message contains the status of a server i.e. CPU and memory usage, of the discovery servers and the status will be marked as **RUNNING**. The monitor maintains all this information in the form of a status table. The status table will be used by the load balancer to distribute the requests accordingly.

Space Invoker

The *Space Invoker* receives sub goal templates $\sum_i^n \mathcal{T}_g^i \equiv \mathcal{T}_\mathcal{G}$ from the parser. For each $\sum_i^n \mathcal{T}_g^i \equiv \mathcal{T}_\mathcal{G}$ it calls the semantic shared space to read and write service description. For each result found for a particular $\sum_i^n \mathcal{T}_g^i \equiv \mathcal{T}_\mathcal{G}$, it forms a (sub) result template or (sub)graph and passes it to the *Result Aggregator*.

Result Filter

This component filters out irrelevant results provided by *Result Aggregator*. It matches the result graphs with the goal graphs and the results are prioritized according to the level of match between them.

Semantic Shared Space

This component acts as a registry for the shared spaces and maintains a mapping between the shared spaces, their IP Addresse(s)(if mirroring is done for that server) and the status of the shared spaces. It provides an interface which accepts a $\sum_i^n \mathcal{T}_g^i \equiv \mathcal{T}_\mathcal{G}$, looks into virtual space and returns the result (sub)graph fulfilling the (sub)goal.

Load Distributor

It works closely with *Monitor* component and redirects the incoming requests to other discovery systems in the network. Here we have used the *Hardware Load Balancing* [1] strategy. The advantages of hardware load balancing are server affinity and high availability. The load distributor acts as a virtual server (also referred to as vserver or VIP) which, in turn, consists of an IP address and port. This virtual server is bound to a number of physical services running on the physical servers in the Semantic shared space. These physical services contain the physical server's *IP address* and *port*.

5.2 Component Interaction

In this section, we describe the working of the whole system right from receiving goal template $\mathcal{T}_\mathcal{G}$ from the client

[1]http://www.onjava.com/pub/a/onjava/2001/09/26/load.html/page=2

to providing the final results. Through the *Access interface* a client provides goal template $\mathcal{T}_\mathcal{G}$ to the *Communication Manager* which validates the goal template $\mathcal{T}_\mathcal{G}$ and if no problems are found the goal template $\mathcal{T}_\mathcal{G}$ is passed to the *Load Distributor*. If validation fails then an error is raised. The *Load Distributor* queries the *Monitor* to look for the availability of a discovery engine. If a discovery engine is available to take a request, *Load Distributor* passes the goal template $\mathcal{T}_\mathcal{G}$ to that engine, else the goal template $\mathcal{T}_\mathcal{G}$ is queued with the *Load Distributor*.

Once the goal template $\mathcal{T}_\mathcal{G}$ is received by the discovery engine, it is parsed by the *Parser* and broken down into sub goal templates $\sum_i^n \mathcal{T}_g^i \equiv \mathcal{T}_\mathcal{G}$, if possible. These sub-goal templates are passed to the *Space Invoker*, for each sub goal the semantic shared space is queried. The semantic shared space looks up into the virtual space and if it finds an appropriate match, returns it in the form of result template or (sub)graph $\mathcal{T}_r$. These result graphs $\mathcal{T}_r$ for each sub goal $\mathcal{T}_g$ are passed to the *Result Aggregator* which aggregates the result graphs $\mathcal{T}_r$ into various possible result graphs $\{\mathcal{T}_\mathcal{R}\}$. This set is given to the *Filter* component, which employs graph matching algorithm on each member of result graph set $\{\mathcal{T}_\mathcal{R}\}$ with the template goal $\mathcal{T}_\mathcal{G}$ and prioritises them according to the matching result.

6. SPACE DISTRIBUTION

The space distribution is an essential aspect of this discovery mechanism. It is needed for creating virtual network of related service descriptions. That is a space contains descriptions of related services. In addition, the distribution mechanism should be capable of distributing queries to the right space in a scalable manner.

6.1 General Space Distribution

The semantic shared space evolves dynamically as service descriptions are stored and removed from it. A new service description is stored in a shared space where similar descriptions are already stored. The similarity is evaluated based on the ontology used to describe functionalities of a Web service. In this state, we use *namespace* together with ontology matching presented in 6.2, for the purpose of identifying similarity between service descriptions.

For the distribution of spaces, multiple mechanism exist. However, to support scalability requirements peer-to-peer mechanisms are deemed suitable for this purpose. By exploiting the *fairness algorithm* [23] service descriptions are distributed over the network so as to balance the network load. A sub-goal $\mathcal{T}_g$ is stored in the shared space that already has service descriptions defined using the same Ontology as in a sub-goal $\mathcal{T}_g$.

6.2 Similarity function

The similarity function, $\mathcal{F}(v_i, v_j)$, implements an algorithm that computes the semantic similarity of any two ontologies. These ontologies are the conceptualization of the data sets related with any two peers of the network, being represented as hierarchies of concepts. In a concept hierarchy one non-leaf concept is a generalization of any of its child nodes. As we move deeper in the concept hierarchy the concepts become more specialized.

Two ontologies are considered similar if their structure and concepts are similar. The process for evaluating similarity between ontology starts with syntactic matching followed by semantic matching matching. In the syntactic matching process, if two concepts ($\mathcal{C}_1$ and $\mathcal{C}_2$) are not same, WordNet is used to try a match between their synonyms. The algorithm, as defined below, checks the matching of leaf concepts from $\mathcal{O}_2$ against every concept from $\mathcal{O}_1$ recursively starting from the leaf concepts. It mainly goes through the following three steps:

a. Initially, the match between the leaf concepts of both the ontologies are checked.

b. If the concept from $\mathcal{O}_2$ does not match with the leaf concept of $\mathcal{O}_1$ then, move to the concept at higher level in $\mathcal{O}_1$ and compare with the concept of $\mathcal{O}_2$, until a either a match is found or the root of $\mathcal{O}_1$ has been reached.

c. If the concept of $\mathcal{O}_2$ does not match with any of the concepts in $\mathcal{O}_1$, then a concept at a higher level is selected from $\mathcal{O}_2$, and compared with the leaf of $\mathcal{O}_1$, and then repeat step b until the match is found between the concepts or the root is reached.

If at any point in the hierarchy the similarity is found, it is important to consider the semantics of the domain, whether the two concepts belong to the same domain of interest. To achieve this, the parents of the concept are matched. If the parents match then it is said that the two concepts have similarity, Otherwise it can be said that these concept belong to different domains and hence have no similarity. This is useful in the situation where there might be similar concept names used in more domains.

In order to make semantic matching reliable it is important to get a feed back from the user. The matched concepts are returned to the user to verify whether this match is acceptable, i.e they are semantically similar. On user's acceptance a *concept similarity factor* is computed for each pair of concepts ($\mathcal{C}_1$, $\mathcal{C}_2$), where $\mathcal{C}_1 \in \mathcal{O}_1$ and $\mathcal{C}_2 \in \mathcal{O}_2$ which is calculated as follows:

$$sim(\mathcal{C}_1, \mathcal{C}_2) = \begin{cases} w_1 * w_2, & \text{if } \mathcal{C}_1 \text{ and } \mathcal{C}_2 \text{ represent the same concept;} \\ 0.0, & \text{otherwise.} \end{cases}$$

Where w_1, and w_2 are the weights of concept $\mathcal{C}_1$ and $\mathcal{C}_2$ respectively.

Weights are assigned during the matching process, depending on the level where the match for the concept is found. The leaf concepts being the most specialized are assigned with a weight value 1. Once the match between two concepts is found the weight calculation takes place and it is assigned to the leaf concept for which the similarity was to be found. As we move up in the hierarchy, the concept nodes get a decreased weight computed by the following formula: $\mathcal{W}(\mathcal{C}) = \frac{1}{\mathcal{D}_t} * \mathcal{D}_\mathcal{C}$, where $\mathcal{D}_t$ is the depth of the hierarchy and $\mathcal{D}_\mathcal{C}$ is distance from the root to the concept $\mathcal{C}$ in the ontology hierarchy.

Once the similarity factor for all the leave concepts are computed for O2, the overall similarity of the two ontologies is computed as follows:

$$\mathbf{similarity}(\mathcal{O}_\mathbf{1}, \mathcal{O}_\mathbf{2}) = \frac{\sum_{i=1}^{n} sim(\mathcal{C}_i)}{n}$$

The objective of using this similarity function is not to provide fine grain ontology matching of concept of the ontologies but to identify the domain they address. The evaluation of algorithm using the Ontology matching evaluation [2], showed very satisfactory results. For brevity the evaluation test were similar to the one performed in Ontology matching evaluation, using ontologies from different domains. Similarity value showed high results for ontology belonging to similar domains whereas ontology irrelevant gave a very low similarity value. Hence possible to identify the threshold value after performing the evaluation.

7. RUNNING EXAMPLE

For the purpose of illustrating the functionality and usage scenario of the proposed architecture the following information encoded by named graph syntax is used.

Using the application scenario presented in the introduction section. Our application acts as a broker. The request is forwarded for discovery to the closest matching domain. Using this scenario there are transportation, postal service, vehicle sale and book seller service providers. Each of the service provider is grouped into based on the domain they support.

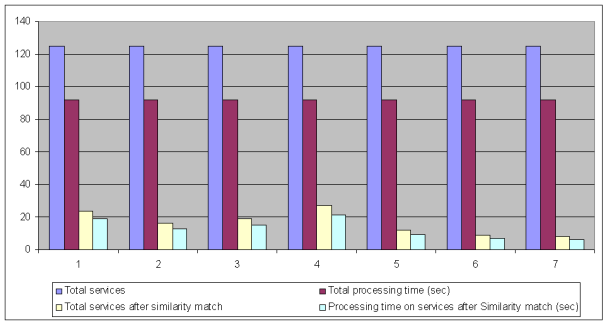
```
@prefix rdf: <http://www.w3.org/1999/02/22−rdf−syntax−ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf−schema> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix ex: <http://www.deri.org/vocabulary#> .
@prefix : <http://www.deri.org/institute/> .

:G1 {
  :AERLINGUS ex:hasName "aerlingus"^^xsd:string.
  :AERLINGUS ex:homepage <http://www.aerlingus.com>.
  :AERLINGUS ex:hasService :IRETOASIA.
  :AERLINGUS ex:hasService :CARGO.
  :AERLINGUS ex:hasService :EUTOASIA.
  }
:G2{
  :IRETOASIA ex:hasName "Ireland to Asia reserve ticket service"^^
      xsd:string.
  :IRETOASIA ex:from :Ireland.
  :IRETOASIA ex:to :ASIA.
  }
:G3{
  :CARGO ex:hasName "EU to US cargo service"^^xsd:string.
  :CARGO ex:from :EU.
  :CARGO ex:to :USA.
  }
:G4{
  :EUTOASIA ex:hasName "EU to Asia reserve ticket service"^^xsd:
      string.
  :EUTOASIA ex:from :EU.
  :EUTOASIA ex:to :ASIA.
  }
:G5{
  :Ireland ex:hasName "Ireland"^^xsd:string.
  :Ireland ex:city "Shannon"^^xsd:string.
  :Ireland ex:city "Dublin"^^xsd:string.
  :Ireland ex:city "Galway"^^xsd:string.
  }
:G6{
  :ASIA ex:hasName "Asia"^^xsd:string.
  :ASIA ex:city "Katmandu"^^xsd:string.
  :ASIA ex:city "Jakarta"^^xsd:string.
  :ASIA ex:city "Lahore"^^xsd:string.
  }
:G7{
  :EU ex:hasName "EU"^^xsd:string.
  :EU ex:city "Galway"^^xsd:string.
  :EU ex:city "Berlin"^^xsd:string.
  :EU ex:city "Oslo"^^xsd:string.
  :EU ex:city "Stockholm"^^xsd:string.
}
:G8{
  :UK ex:hasName "UK"^^xsd:string.
  :UK ex:city "London"^^xsd:string.
  :UK ex:city "Liverpool"^^xsd:string.
  :UK ex:city "Glasgow"^^xsd:string.
  }
:G9{
  :AFRICA ex:hasName "Africa"^^xsd:string.
  :AFRICA ex:city "Cairo"^^xsd:string.
  :AFRICA ex:city "Durban"^^xsd:string.
  :AFRICA ex:city "Harare"^^xsd:string.
  }
:G10{
  :USA ex:hasName "USA"^^xsd:string.
  :USA ex:city "New York"^^xsd:string.
  :USA ex:city "Chicago"^^xsd:string.
  :USA ex:city "Boston"^^xsd:string.
  :USA ex:city "Florida"^^xsd:string.
}
:G11 {
  :CITYLINK ex:hasName "CityLink"^^xsd:string.
  :CITYLINK ex:homepage <http://www.citylink.ie>.
  :CITYLINK ex:hasService :IRETOASIA.
  :CITYLINK ex:hasService :CARGO.
  }
:G12{
  :IRETOUK ex:hasName "Ireland to UK reserve ticket service"^^xsd:
      string.
  :IRETOUK ex:from :Ireland.
  :IRETOUK ex:to :UK.
  }
:G13{
  :ASIATOAFRICA ex:hasName "ASIA to AFRICA reserve ticket
      service"^^xsd:string.
  :ASIATOAFRICA ex:from :ASIA.
  :ASIATOAFRICA ex:to :AFRICA.
  }
:G14 {
  :AMAZON ex:hasName "Amazon"^^xsd:string.
  :AMAZON ex:homepage <www.amazon.com>.
  :AMAZON ex:hasService :SONYDIGICAM.
  :AMAZON ex:hasService :PANASONICDIGICAM.
  :AMAZON ex:hasService :WILEYPUB.
  }
:G15{
  :HERZ ex:hasName "HERZ"^^xsd:string.
  :HERZ ex:homepage <www2.hertz.co.uk>
  :HERZ ex:hasService :TOYOTACAR.
  }
:G16{
  :SONYDIGICAM ex:hasName "SONY Digital Camera"^^xsd:string.
  :SONYDIGICAM ex:resolution "7.5"^^xsd:decimal.
}
:G17{
  :PANASONICDIGICAM ex:hasName "PANASONIC Digital Camera
      "^^xsd:string.
  :PANASONICDIGICAM ex:resolution "8.5"^^xsd:decimal.
}
:G18{
  :WILEYPUB ex:hasName "WILEY PUBLISHERS"^^xsd:string.
}
:G19{
  :TOYOTACAR ex:hasName "TOYOTA CAR"^^xsd:string.
}
```

Listing 3: Information in Named Graph

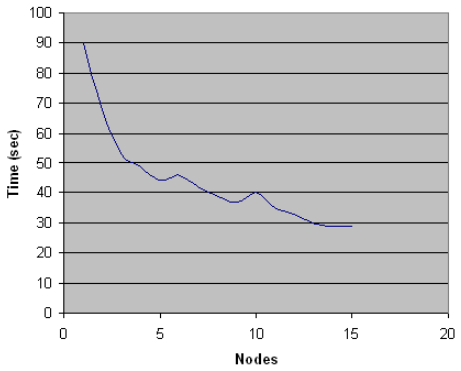
In this example, several named graphs are used to encode information regarding company, services offered, geographical locations and products sold by the companies. Let us illustrate its publication and retrieval in our distributed discovery system.

Publishing or Writing - Initially, this document is presented to the *Communication Manager* through the *Access Manager*. As the syntax is correct it is given to the *Load Distributor* which after consulting *Monitor*, passes it to a distributed system in the network. The received document is then parsed by *Parser* to get 19 different sub graphs (:G1

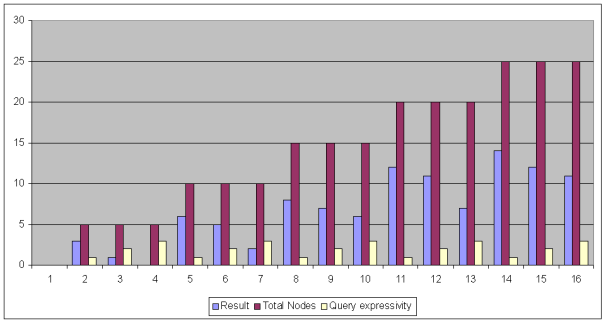
[2]http://www.ontologymatching.org/



(a) Grouped vs Ungrouped



(b) Response Time



(c) Reachability

Figure 3: Evaluation Results

to :G19). The sub graphs :G1, :G11, :G14 and :G15 being similar are stored in one shared space, where as :G2, :G4, :G12 and :G13 are stored in another shared space. Another shared space containing sub graphs :G5, :G6, :G7, :G8, :G9 and :G10 is created. The sub graphs :G16 an :G17 are also placed in the same shared space. The sub graphs :G3, :G18 and :G19 are stored in three different shared spaces. The storage is performed through the *Space Invoker* component. These shared spaces are connected following the Semantic shared space principle as mentioned in Section 4.

Retrieving or Reading - In the event of information retrieval the SPARQL protocol is used for defining the query as shown below. In this example, a user intends to find name and homepage of the companies providing travel service between Europe and Asia.

```
1  PREFIX ex: <http://www.deri.org/vocabulary#> .
2
3  SELECT ?hasName ?homepage WHERE
4    { GRAPH ?anygraph { ?graph
5          ex:hasService ex: EUTOASIA }
```

Listing 4: Query in SPARQL

This request is passed to the *Space Invoker* after its execution through *Communication Manager*, *Parser*, *Monitor*, and *Load Distributor*. The *Space Invoker* invokes Semantic shared space and obtains all the graphs that have information regarding the companies providing travel service between Europe and Asia. This result is sent to the *Result Aggregator* which then aggregates information from related sub-graphs thereby generating a set of possible result graphs. Finally, this set is passed to the *Filter* which discards the information that do not match the original query. The filtered result is then sent to the user.

8. EXPERIMENTS AND EVALUATIONS

A prototype of the proposed architecture is implemented and some experiments are conducted. This prototype is implemented on top of JXTA [3] peer-to-peer infrastructure to enable space distribution. This evaluation was carried out against both the real and synthetic data sets. In total, around 125 Web service descriptions from seven different domains were used. In the experiment, processing time per node as well as process time over the network is noted. In general the semantic similarity, recall, reachability as well as query expressivity is evaluated. The results are shown in the following figures. As seen in Figure 3(a), the processing time of discovering services in case of semantic grouping is significantly lower than that in the ungrouped case. The graph shows six groups. First evaluation shows the processing on one group in which no similarity algorithm is performed. Next groups were created and processing for discovery is performed. It is clear from the graph that there is a significant improvement in processing time. In one group discovery is performed on all the services, when grouped query is only routed to respective group and search space is reduced hence the processing time. This evaluation has been performed on a single node. Next the evaluation is performed on the peer-to-peer network of discovery modules. The result of this evaluation is as shown in Figure 3(b). It shows that the response time decreases significantly with the increase in number of nodes. Using the similarity algorithm the services are distributed between the nodes and results in the decrease in overall response time. Services are distributed amongst the nodes belonging to the same group. Initially discovery evaluation was performed on only one node. While increasing the nodes and resulting semantic groups load of discovery is distributed, thus taking less time for the nodes to respond. Figure 3(c) provides an analysis of query expressivity and node reachability. It gives the indication as the expressivity is increased the precision also increases. In our experimentation, the expressivity of the query varies from one logical to three logical expressions along with the number of nodes varying from 5 to 25 nodes. It is clear from the graph that increasing the expressivity of the query, increases the accuracy of the result. While increasing the nodes the number of responses also increases, which proves the reachability of the architecture.

[3]https://jxta.dev.java.net/

9. RELATED WORKS

There are several approaches proposed in the literature for efficient location of required services. The combination of semantic Web and tuple spaces is proposed in [24–27].

This list however is not exclusive.

Linda-like coordination mechanism for discovery in mobile environment is proposed in [24]. It exploits a flat data structure and extends Linda interfaces with a location parameter λ, expressed in terms of agents or host identifier. Unlike in our approach, it does not incorporate semantics in its data model. In [25], a document discovery mechanism is presented. It realises a forest of trees composed of nodes and documents. These trees in the forest are coordinated by each peer. Its basic access primitive *execute* requires the $\mathcal{F}_D$ and $\mathcal{F}_N$ functions to filter the operated set of documents and nodes. Compared to this system, our approach ensures better recall and result precision.

In [26], support for publishing and discovering WSDL files is provided. It enables discovery of these files and their retrieval from remote machines. Using the Discovery Document Format (which is also an XML grammar), one can send a discovery document to a remote server and, if any SOAP enabled services exist, receive back WSDL description of the services provided. Unlike in our approach, there is a tight coupling between service provider and requester applications. By exploiting tuple space principle we support loose coupling and reference autonomy.

The discovery mechanism presented in [27] shares a 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. Semantic support is the main strength of our architectural solution. Service requesters have to obtain security key from the provider to view the service description which is an undesirable feature of the distributed system. In addition it adds an extra communication overhead.

A P2P web service discovery approach is presented in [28]. They provide web service discovery approach using an enhanced skip graph for semantic description lookup using WSDL-S for describing web services. Although they utilize semantic description for web services and create keywords, but they only use keyword matching for routing and leaving the matchmaking for the second level using Vector Space Model.

10. CONCLUSION

In this paper a distributed semantic service discovery architecture is presented. This architecture is based on semantic shared space principle where spaces are distributed over the Internet. One of the key feature of this architecture is that the spaces are self organized. It allows interaction between distributed applications supporting scalability and fault tolerance. Through evaluation it is seen that grouping semantically similar services reduces both execution as well as response time. With the increase in the number of peers in the network, the recall as well as precision is improved in comparison to that in single repository. In the next phase of this work, it will be extended to support communication between mobile and resource limited devices.

Acknowledgment

The work presented in this paper was supported (in part) by the Líon project supported by Science Foundation Ireland under Grant No. SFI/02/CE1/I131 and (in part) by the EU projects TripCom No. IST-4-027324-STP, SemanticGov No. FP6- IST-4-027517

11. REFERENCES

[1] Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American (2001)

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

[3] Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web Services. Springer-Verlag, Berlin, Heidelberg (2004)

[4] McIlraith, S.A., Son, T.C., Zeng, H.: Semantic Web Services. IEEE Intelligent Systems. Special Issue on the Semantic Web **16**(2) (2001) 46–53

[5] Fensel, D., Bussler, C.: The Web Service Modeling Framework WSMF. Electronic Commerce Research and Applications **1**(2) (2002) 113–137

[6] Roman, D., Lausen, H., Keller, U., eds.: Web Service Modelling Ontoloty (WSMO). WSMO Deliverable, version 1.2 (2005)

[7] Martin, D., Paolucci, M., McIlraith, S., Burstein, M., McDermott, D., McGuinness, D., Parsia, B., Payne, T., Sabou, M., Solanki, M., Srinivasan, N., Sycara, K.: Bringing Semantics to Web Services: The OWL-S Approach. In: Proc. First International Workshop on Semantic Web Services and Web Process Composition, San Diego, California, USA (2004) 26–42

[8] Oundhakar, S., Verma, K., Sivashanmugam, K., Sheth, A., Miller, J.: Discovery of Web Services in a Multi-Ontology and Federated Registry Environment. International Journal of Web Services Research **2**(3) (2005) 1–32

[9] Fensel, D.: Triple Space Computing. DERI Technical Report 2004-05-31 (2004) available at: http://www.deri.org/TR/2004-05-31.

[10] Eugster, P.T., Felber, P.A., Guerraoui, R., Kermarrec, A.M.: The Many Faces of Publish/Subscribe. ACM Computing Surveys **35**(2) (2003) 114–131

[11] Leymann, F.: A Practioners Approach to Database Federation. In: Proc. 4th Workshop on Federated Databases, Berlin, Germany (1999)

[12] Microsystems, S.: JavaSpace Specification (1998)

[13] Wyckoff, P., McLaughry, S., Lehman, T., Ford, D.: T Spaces. IBM Systems Journal **37**(3) (1998) 454–474

[14] Sapkota, B., Kilgarriff, E., Bussler, C.: Role of Triple Space Computing in Semantic Web Services. In: Proc. APWEB 2006, Harbin, China (2006) 714–719

[15] Klyne, G., Carroll, J., eds.: Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation (2004)

[16] Hayes, P., ed.: RDF Semantics. W3C Recommendation (2004)

[17] McGuinness, D.L., van Harmelen, F., eds.: OWL Web Ontology Language. W3C Recommendation (2004)

[18] de Bruijn, J., ed.: Web Service Modelling Language (WSML). WSML Deliverable, version 16.1 (2005)

[19] Carroll, J.J., Bizer, C., Hayes, P., Stickler, P.: Named graphs. Journal of Web Semantics **3**(4) (2005) 247–267

[20] Bussler, C.: A Minimal Triple Space Computing Architecture. In: Proc. 2nd WSMO Implementation Workshop, Innsbruck, Austria (2005)

[21] Prud'hommeaux, E., Seaborne, A., eds.: SPARQL Query Language for RDF. W3C Candidate Recommendation (2006)

[22] Sapkota, B., Vasiliu, L., Toma, I., Roman, D., Bussler, C.: Peer-to-Peer Technology usage in Web Service discovery and Matchmaking. In: Proc. WISE 05, New York, USA (2005) 418–425

[23] Jain, R., Chiu, D., Hawe, W.: A Quantitative Measure Of Fairness And Discrimination For Resource Allocation In Shared Computer Systems. (DEC Technical Report TR-301)

[24] Picco, G.P., Murphy, A., Roman, G.: LIME: Linda Meets Mobility. In: International Conference on Software Engineering, Los Angeles, USA (1999) 368–377

[25] Cugola, G., Picco, G.: PeerWare: Core Middleware Support for Peer-To-Peer and Mobile Systems. Technical report, politecnico di milano (2001)

[26] Tomasic, A., Raschid, L., Valduriez, P.: Scaling Heterogeneous Distributed Databases and the Design of DISCO. IEEE Transactions on Knowledge and Data Engineering archive **10**(5) (1998) 808–823

[27] Hsiao, H.C., King, C.T.: Neuron - A Wide-Area Service Discovery Infrastructure. In: International Conference on Parallel Processing (ICPP'02), Vancouver, Canada (2002) 455–462

[28] Zhou, G., Jianjun, Y., Chen, R., Zhang, H.: Scalable Web Service Discovery on P2P Overlay Network. In: International Conference on Services Computing (SCC'07), Salt Lake City, Utah, USA (2007) 122–129