REGISTERING AND DISCOVERING SEMANTIC WEB SERVICES IN A FEDERATED DIRECTORY SYSTEM

Michael Schumacher
University of Applied Sciences Western Switzerland, CH-3960 Sierre, Switzerland
michael.schumacher@hevs.ch

Alexandre de Oliveira e Sousa, Ion Constantinescu, Tim van Pelt and Boi Faltings
Ecole Polytechnique Fédérale de Lausanne (EPFL), CH-1015 Lausanne, Switzerland
firstname.lastname@epfl.ch

Abstract

This paper presents a federated directory system, which allows registration and discovery of semantic web services. The typical use of this directory system is in a context where pervasive ubiquitous business application services should be flexibly coordinated and pervasively provided to the mobile user by intelligent agents in dynamically changing environments. The system has been modeled, designed and implemented as a backbone directory system to be searched by an infrastructure made up by such kind of agents coordinating semantic web services. The system is modeled as a federation: directory services form its atomic units, and the federation emerges from the registration of directory services in other directory services. Directories are virtual clusters of service entries stored in one or more directory services. To create the topology, policies are defined on all possible operations to be called on directories. For instance, they allow for routed registration and selective access to directories. WSDir has been applied as a backbone in the trials of an ehealth emergency application.

Keywords: Semantic Web services, federated directories, mobile computing, intelligent agents

1 INTRODUCTION

This paper[1] presents a new federated directory (or registry) system called WSDir[2][8], which allows registration and discovery of OWL-S semantic web services [6]. Its main functionality is to let heterogeneous semantical service descriptions be registered and searched, including using a semantic matchmaker.

The basic vision is that ubiquitous application services are flexibly coordinated and pervasively provided to the mobile users by intelligent agents in dynamically changing contexts of open, large-scale, pervasive environments. In such dynamical pervasive context, a directory system is a key component to locate and discover specific application services.

Our directory system is used in the CASCOM platform[3]. Its essential approach is the innovative combination of intelligent agent technology, semantic Web services, distributed directories, peer-to-peer, and mobile computing for intelligent peer-to-peer (IP2P) service environments. The services are provided by software agents exploiting the co-ordination infrastructure to efficiently operate in highly dynamic environments.

We modeled, designed and implemented WSDir as a backbone directory system to be searched by an infrastructure made up by such kind of agents coordinating web services. This agent infrastructure therefore is deployed on mobile users: it queries our directory system for OWL-S service descriptions, composes them to achieve a target higher functionality and executes them. WSDir has also been used in the trials of an ehealth emergency application.

We followed specific requirements for designing WSDir: i) it should have itself a Web service interface to be universally invoked; ii) it should be distributed; iii) the construction of the network should induce minimal overhead and should be scalable; also, the network should be robust to changes in topology and the number of interactions with the system; iv) the directory should allow services to be registered in a very dynamic way, including lease times.

These requirements lead us to model WSDir as a

[1]This work has been supported in part by the European Commission under the project grant FP6-IST-511632-CASCOM.
[2]This paper is an extended version of [8]. Furthermore, the source code and the documentation of WSDir are available on wsdir.epfl.ch.
[3]http://www.ist-cascom.org

federation: directory services form its atomic units, and the federation emerges from the registration of directory services in other directory services. Directories are virtual clusters of service entries stored in one or more directory services. To create the topology, policies are defined on all possible operations to be called on directories. For instance, they allow for routed registration and selective access to directories.

The paper is organized as follows. Sec. 2 sets the context of directory services in pervasive environments. In Sec. 3, we explain the service entries of semantic web services that can be stored in WSDir. Sections 4 to 7 explain WSDir by presenting *directories*, *directory services*, *directory operations*, and *policies*. In Sec. 8, we give a concrete network architecture example based on our federated directory system. Sections 9, 10 and 11 discuss respectively usability, vulnerability and performance issues of WSDir. After referring related work in Sec. 12, we conclude the paper in Sec. 13.

2 SERVICE COORDINATION IN MOBILE COMPUTING ENVIRONMENTS

In this section, we explain the context and the motivation of the use of our distributed directory system. As stated in the introduction, our driving vision is that of the CASCOM infrastructure, where ubiquitous business application services are flexibly coordinated and pervasively provided to mobile workers and users by intelligent agents in dynamically changing contexts of open, large-scale, and pervasive environments.

Its approach is a combination of agent technology, Semantic Web service coordination, P2P, and mobile computing for intelligent peer-to-peer (IP2P) mobile service environments. IP2P environments are extensions to conventional P2P architectures with components for mobile and ad hoc computing, wireless communications, and a broad range of pervasive devices. Basic IP2P facilities come as web services, while their reliable, task-oriented, resource-bounded, and adaptive co-ordination-on-the-fly characteristics call for agent-based software technology. One has to guarantee a secure spread of service requests across multiple transmission infrastructures and ensure the trustworthiness of services that may involve a variety of providers. The services of the infrastructure are provided by peer software agents exploiting the co-ordination infrastructure to efficiently operate in dynamic environments. The IP2P infrastructure includes efficient communication means, support for context-aware adaptation techniques, as well as dynamic and secure service discovery and composition planning.

Given that pervasive IP2P environments assume that users are providing services to other users, it is essential to have an efficient and dynamic mean to store and search those application services. WSDir fulfills exactly this functionality with a distributed infrastructure for storing and searching service descriptions.

3 SERVICE ENTRIES

We describe services using the Web Ontology Language for web services, OWL-S [6]. The directory system stores all descriptions translating from the OWL-S description towards a representation described in FIPA SL0 [4] that will include the original service description. Additional fields contain information that is taken from the service description in OWL-S.

Internally, WSDir stores service entries in the FIPA SL0 description language. The use of SL0 has a very important advantage: it allows to store any kind of service descriptions, and not only OWL-S. To add a new kind of services, we only have to define a new translator towards our SL0 description.

The internal service representation contains a subset of the information provided in the original description. This information can be used to find matching services in the directory. In addition, the original service description in OWL-S is stored in a separate slot. This field is used to retrieve the original description, e.g., to retrieve the grounding(s) of a service at service execution.

A service entry in WSDir contains the following information: i) *ServiceCategories*, and entry in some ontology or taxonomy of services; ii) *ServiceProfileURIs*, a set of profile URIs that point to an externally stored, but (web-)retrievable service profile; iii) *ServiceProcessURI*, a process URI defined in the service description (can be empty); iv) *ServiceGroundings*, a set of full-text service groundings (can be empty); v) and *OWLSServiceDescription*, an original OWL-S service description as a full-text entry.

4 DIRECTORIES

A directory comprises a set of service entries which are managed by a collection of one or more directory services. All service entries, including directory service entries, are registered at a directory service as belonging to a specific directory. Such, directory services can form an arbitrary organizational structure (peer-to-peer, hierarchy etc.). A directory can contain other directories, and a directory is supported by one or more directory services. This is used to characterize directories by

two different types of interactions. In *Client-Directory* interactions, clients are registering, deregistering, and querying the directory interact with directory services supporting the directory. They may or may not have any idea about the internals of the directory. In *Director-Directory* interactions, directory services supporting the directories interact with one another to perform the internal management of the directory (data propagation, federated queries, managing the membership of the directory service group managing the directory).

The first interaction style (*Client-Directory*) is part of the base for the creation and maintenance of a *domain directory*. The second interaction style (*Directory-Directory*) is the base for the creation and maintenance of a network directory. As directories can be used to support other directories they are seen as organizational structures. These concepts will be elaborated and exemplified in Sec. 8.1 on network topology.

5 DIRECTORY SERVICES

Directory services provide a Web service interface to a repository that holds service entries. The service entries in this store are all registered as belonging to a certain directory. The directory service forms the atomic unit of the directory federation.

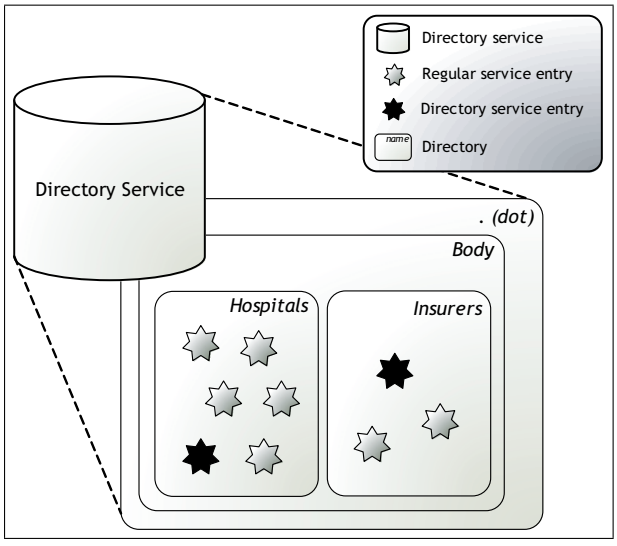

Figure 1: Directory system concepts

The directory service allows clients to register, deregister, modify and search registrations in its repository. These registrations include service descriptions of services offered by clients as well as profiles of other directory service. By registering directory services in other directory service stores, the system becomes federated. Figure 1 visually summarizes the relationship between service entries, directories and directory services. In the illustration, the directory service holds regular service entries and a directory service entry belonging to a Hospitals directory as well as entries belonging to an Insurers directory. Both directories are contained in the Body directory, which in turn is contained in the all-encompassing "." directory.

6 DIRECTORY OPERATIONS

The directory is able to handle five types of operations. The *register* operation enables a client to register a service description entry into a *directory* for a time period given by a *lease-time*. The *directory-service* can return a *redirect* message pointing to another directory where the client could try to register its object. The *deregister* operation de-registers a service that previously has been registered identified. The *modify* operation allows modifying a registered service. The *get-profile* requests meta-information on a directory service such as its name and the policies governing the operations.

The *search* operation looks in the directory for services that match a template. The request can possibly be forwarded to supporting directories, depending on the implemented search policy at the directory. As the internal service descriptions are expressed as SL0 expressions, the structured element used in the operation is also an SL0 expression. *Search-constraints* can be specified in order to restrain the search.

There are three types of search requests: *abstract* (search by abstract services), *grounding* (search by service groundings) and *matchmaker* (search by using a semantic matchmaker). The matchmaker type is highly dependent on the matchmaker module used by WSDir (OWLS-MX [5] for the CASCOM project).

A *max-time* specifies a deadline by which the constrained search should return the results. A *max-depth* specifies the maximum depth of propagation of the search to federated directories. A *max-results* element specifies the maximum number of results to be returned.

7 POLICIES

Directory services employ *directory policies* to regulate the operation of directories. Policies are defined per directory service in the *directory service profile* and determine the behaviour of a specific directory. Two types of policies can be distinguished. *Pro-active policies* are typically used for internal management of the directories. A policy may be attached to a directory to establish the number of times per hour data is propagated within the directory, how often old entries are removed etc. *Re-*

active policies assign a behaviour to combinations of directories and operations. The policies are executed whenever a bound operation is called. They are defined as a triple: *(directory name, operation, policy)*. Policies can also be applied to the default directory named "*" which matches all directories that don't have a policy explicitly assigned.

From the consequent application of policies, the network topology emerges. Policies can for example define how much entries can be registered per directory, which directories can be searched by which clients, and which types of services will be accepted. Each directory service can define its own policies or use one of the predefined policies. The only requirement on the part of a policy is that it can be executed.

A straightforward example of a pre-defined policy is the *child/sibling* policy: this policy forwards all operations to both the known children (directory services registered in the service store) as well as its known siblings. The list of siblings is obtained by querying the parent directory service at which it has registered itself.

Figure 2 shows an example of the reactive policies that are assigned to the various directories this directory service supports. In the illustration, when a client calls the search operation on the "Hospitals" directory, a policy called "secauth" will be applied. Such a policy could for example require the client to authenticate itself before it is allowed to query the directory.

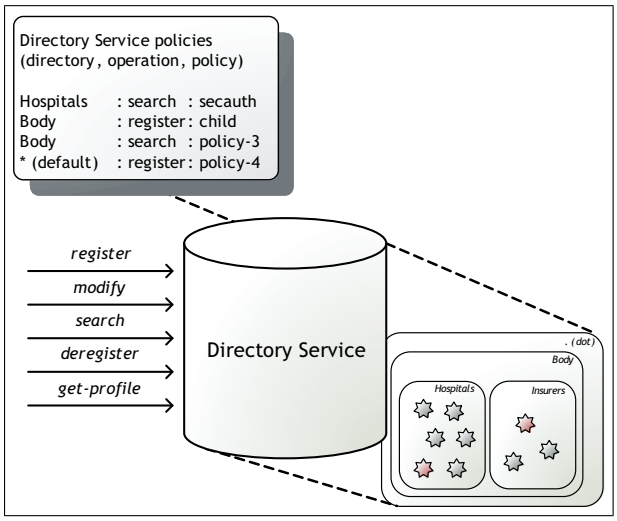

Figure 2: Example use of policies

8 AN EXAMPLE OF A NETWORK ARCHITECTURE

WSDir allows to setup flexible distributed directory systems, especially thanks to the mechanism of policies. We present here a specific application of WSDir to build a network of directory services which are modelled as a virtual tree with multiple roots. This topology has been applied in the trial of a medical emergency usecase scenario in the CASCOM project.

8.1 Network Topology

The nodes of the tree are made up by the individual directory services. This hierarchical structure with multiple entry points effectuates no replication or data caching within the directory: each directory service is responsible for registrations made in its local store. Since there is no replication, directory services will forward queries to other directory services. Query messages are checked for duplication: a given directory service will handle only the first of several identical query messages from several sources and discard the others while returning the appropriate failure message. Furthermore, dentical results may be returned by one or more directory services for a single query. This enhances the robustness of the federation at the cost of shifting the burden of filtering the results to the client.

We distinguish two types of directories: network directories and domain directories. *Network directories* are a reserved set of directories that are used for the construction of the network. In a directory federation, we can distinguish three different network directories:

- *Hidden* network directory: the directory service that forms the root of the federation by registering the top-level nodes of the network. Neither the directory service nor its registered services will be visible to the other nodes in the federation.
- *Top* network directory: visible to the network as being one of the roots of the federation multi-rooted tree. A directory service with this role could typically serve as a bootstrap service to leaf directory services. These services constitute the Top network directory.
- *Body* network directory: regular directory services that form the body of the multi-rooted tree. These directory services provide the interface to the directories that will contain most of the service registrations in the network.

Domain directories emerge from the registrations of service descriptions at the directory services that make up the directory system. By definition, domain directories are contained in the "Body Members" directory.

Hereafter, we explain the network directories in more detail. The above-mentioned roles and their place in a WSDir federation topology are depicted in Fig. 3.

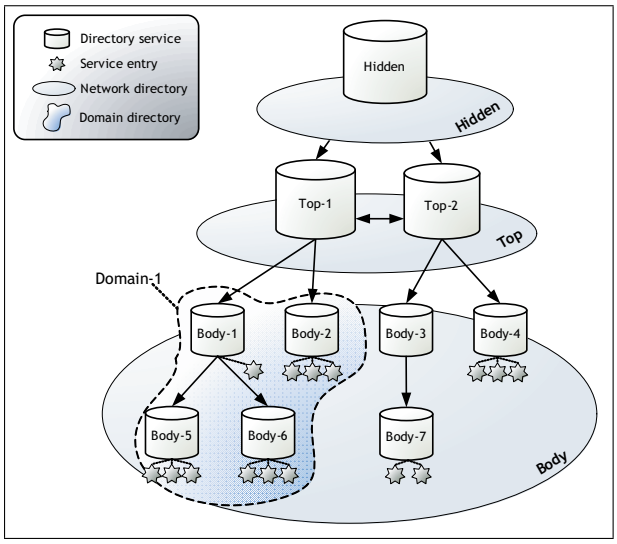

Figure 3: Network Topology

8.1.1 Hidden network directory

The Hidden directory service node responds only to requests coming from Top directory services and exclusively regarding the ”Top Members” directory. For that it holds authentication information regarding a pre-configured list of possible top-level nodes and uses this information together with information in the identification information field of the requests. Search queries are not propagated to other directory services. The location of the hidden directory service node is pre-defined.

The existence of this domain ensures that directory services belonging to the ”Top Members” directory know of their respective existence and such makes sure that every node in the network can be reached if needed. The hidden directory service is only used for bootstrapping of the top member directory services. After the system has been initialized, the hidden directory service will only be used by the top member directory services to poll to see whether new directory services have been added to the ”Top” network directory. Thus, queries, registrations and other requests do not go through the hidden directory service, but will be directed at a top or body member directory service.

8.1.2 Top network directory

Upon start-up the Top directory services join the network by registering their directory-service-profiles inside the ”Top Members” directory of the Hidden directory service. Also they keep track of other Top directory services currently members of the ”Top Members” directory by continuously polling the ”Top Members” directory of the Hidden directory service for directory-service-profiles entries. In the case that the Hidden directory service fails the Top directory services should continue to use the last retrieved membership information until the Hidden directory service will be back online. In terms of response to registration requests from clients, a Top directory service allows only for the registration of directory-service-profile entries inside the ”Body Members” directory. Once the number of registrations that are hold locally goes over a given threshold, the Top directory service returns redirect messages pointing requestors. Normal directory services directly registered with the current directory service. For any other kind of registration requests, the Top directory services will issue redirect responses pointing at Normal directory services registered with the current directory services or other Top directory services that might be more appropriate for use. Top directory services will respond to all search requests by first trying to fulfil them locally and in the case that more results can be returned (the value of the max-results parameter in the search-constraints object has not been reached yet) it will forward the query to all other Top directory services members of the ”Top Members” network directory. For determining the other Top directory services members of the ”Top Members” directory, the information from the last successful polling of the Hidden directory service will be used. Top directory services will respond with a failure to all other kinds of requests.

8.1.3 Body network directory

At start-up, a Body directory service will try to register its directory-service-profile in the ”Body Members” directory of a Top directory service randomly picked from a pre-configured list of Top directory services. If the Top directory service cannot be reached another one is randomly picked until either the joining procedure (see next) succeeds or the list is exhausted. In the latter case the directory service will report a join failure. The directory service will follow redirect responses until the entry is successfully registered with a directory service (either Top or Body). Upon failure of the directory service used for registration the current directory service will sleep for a random time period and after than will re-initiate the initial join procedure.

For other Body directory services that try to register directory-service-profile entries inside the ”Body Members” directory a Body directory service will act as a Top directory service: once the number of registrations that are hold locally goes over a given threshold the Body directory service will return redirect messages pointing requestors to child Body directory services directly registered with the current directory service.

A Body directory service will respond positively to all other requests. In particular it will forward search queries for which it could return more results than locally available to directory services locally registered in the ”Body Members” directory.

8.2 Network Construction

At boot time, the directory makes use of a pre-defined network configuration to create a network topology. The configuration specifies management and data relations between members of the network.

Some of the network nodes might have fixed well-known addresses in order to serve as bootstrap hosts for other directory services. Depending on their role, different parts of the network are visible to bootstrapping directory services.

As mentioned before, in a typical setting, the node at the highest level will be hidden to all nodes not belonging to the ”Top Members” directory.

The process of directory service registration is equivalent to the process of registering regular service entries. Directory services are registered invoking the same register method as is used for registering regular services.

WSDir employs a set of pre-defined pro-active policies that will allow to construct topologies. Figure 4 gives an overview of the pre-defined policies and their hierarchy. We do not present here the details of each policy. However, Fig. 5 shows the policies applied to construct the basic network topology . Per network directory, the set of policies in place is listed.

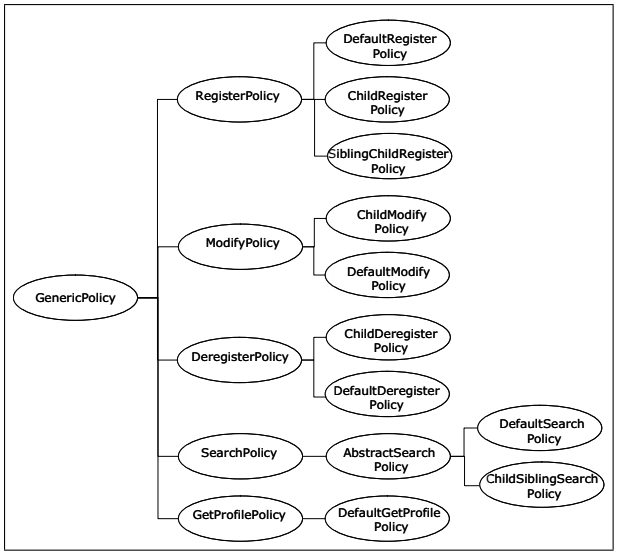

Figure 4: Predefined Policy Tree

For example, a registration request directed at a directory belonging to the ”Hidden” network directory triggers the application of the *ChildRegisterPolicy*. The service registration request is forwarded to its known children, which themselves apply (by default) the *ChildSiblingRegisterPolicy*. This in turn selects the least loaded directory service among its children and among its siblings to put the service entry in its store.

The procedure for a *search* operation is similar. Requests directed at a directory service either belonging to the ”Hidden” network directory or to the ”Top” network directory will forward the search request to its registered children and its known siblings. The directory service instances belonging to the ”Body” network directory apply the *DefaultSearchPolicy*, only searching their local store.

For the *modify*, *deregister* and *get-profile* operation, the policies that are assigned to the operations also depend on the network directory the directory service instance belongs to.

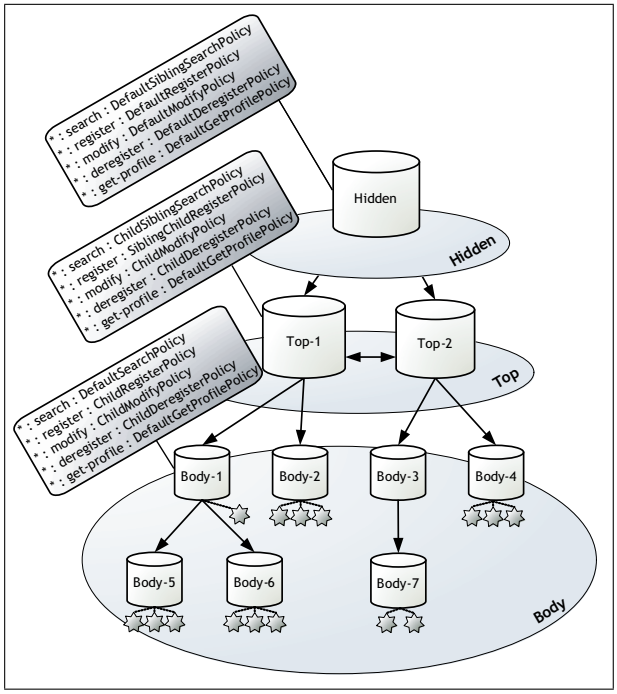

Figure 5: Policies in the Network Topology

8.3 Examples of Network Interactions

The following section describes one example of the policy-governed query operation of the network of directories as presented previously. The visible network is formed from a number of ”well-known” top nodes (Top-1, Top-2, Top-3) with a fixed name and transport address (but which can possibly fail) and an arbitrary number of leaf nodes which are organized in a tree topology with one of the top nodes as root.

Query resolution is illustrated in Fig. 6: a client

issues a search request for service profiles matching a template in the *Hospital* domain directory. i) First, the client issuing the query randomly selects one of the top level nodes (*Top-1*, *Top-2*, *Top-3*) (in this example, *Top-2* is picked). ii) The *Top-2* directory service forwards then the query to its siblings. iii) This allows directory services that have an entry for the *Hospital* domain directory in their service profile to propagate the query down. iv) Then, to guarantee full query resolution, the query is forwarded to all directory services that are known to store entries for the *Hospital* domain directory. v) Finally, upon finding results, the nodes holding the results will send a message back (depicted by "R=x" in the figure, where x denotes the number of matched services) to the directory service it was queried by, until it reaches the original requester. In this case, the matching service profiles in the directory services supporting the *Hospital* domain directory will be returned.

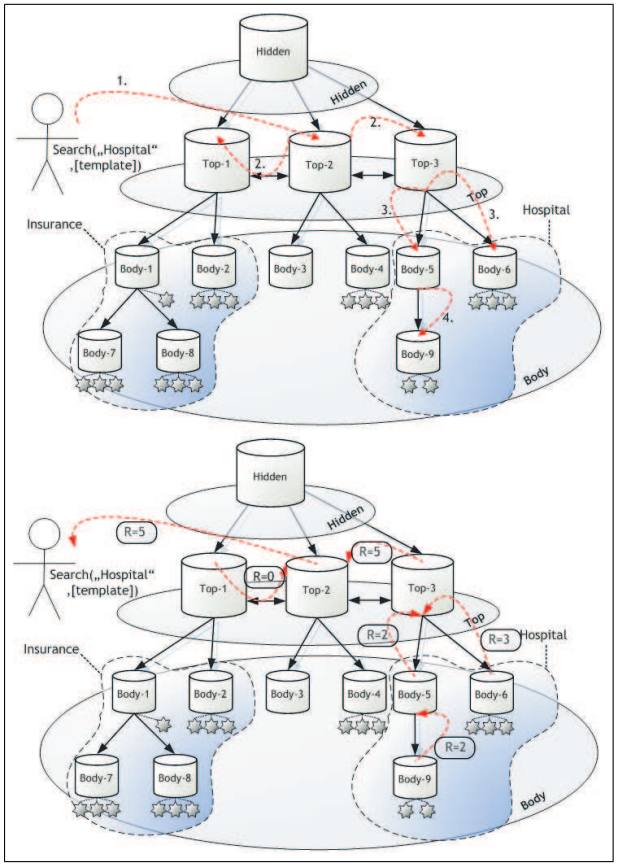

Figure 6: Query Resolution

9 USABILITY

For every instance of a WSDir's Directory Service, the user must write its own configuration file. This configuration file is accessed and read by the Directory Service during its starting procedure. The name of this file must be explicitly written in a file web.xml of the Directory Service.

The syntax of this file is based on FIPA SLO. It contains all the neccessary information for the Directory Service to create its directories, associate the policies and start building a predefined network topology with other Directory Services. The user must specify the following information: i) name and address of the Directory Service; ii) name(s), address(es) and credentials of the Directory Services it should register in; iii) name of the directories it manages; iv) name of the policies that applies to directories for each operation.

Another issue regarding the WSDir's usability is monitoring its's run time activity. There are currently two ways to do this. The first one is to use a Java client which enables a human user to browse through the Directory Services and their directories. Directories and services entries are displayed in a visual tree. The user can fold/unfold directories and check which services are currently registered in a Directory Service. The second way to monitor WSDir's activity is to deploy a servlet on the same server where an instance of a Directory Service is running. Depending on this Directory Service's configuration, a user will be able to access a web page that displays a set of logs of registration requests made on it. Thus, the user can check whether his requests (registration, modification or remove) were successfully executed.

In either way, no performance information nor disfunction messages are being displayed to the user monitoring WSDir's activity. The user is then leaded to check the logs files if something wrong happened. Monitoring Directory Services performance at run time as well as errors would be a major contribution to WSDIR's usability.

On the other hand, once a topology has been decided and the configuration files have been written correctly, it is very easy to launch a federation. There exists a Java class (Startservices) that takes an ordered list of Directory Service addresses and automatically launch all of them. The user can also take advantage of another graphical tool that enables him to directly send a request to a specific Directory Service.

10 VULNERABILITY

In this section, we discuss two major issues in WSDir's vulnerability. The first one concerns server failures and breakdowns. The second one concerns more the security restrictions and users rights. In both cases, WSDir copes with those issues by using specific mechanisms.

WSDir uses its loosely coupled directories and a data backup system to efficiently handle breakdowns. It implements an authentication mechanisms to identify the clients sending incoming requests.

10.1 Breakdowns

In the CASCOM project, we have used the network topology presented in 8.1, where the federation is structured in three Network layers: the Hidden layer, the Top layer and the Body layer. Although all Directory Services, regardless from which Network they belong to, can operate all requests, only those situated in the Top layer are accessed by the CASCOM's Discovery Agents. As each Network layer plays a specific role in WSDir's Federation, three breakdown scenario are discussed. Figure 7 illustrates the accessible Service Descriptions stored in a WSDir's Federation when all the Directory Services are running correctly. All descriptions can be accessed by a Client (the group of accessible Directory Services is defined by the quadratic border).

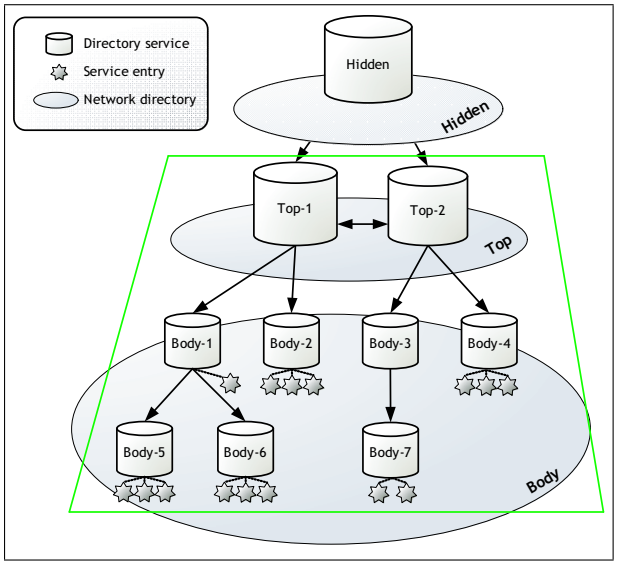

Figure 7: A WSDir Federation with all the Directory Services from each Network layer is working correctly. The quadratic border defines the group of currently accessible Service Descriptions stored in the Federation.

In a first failure scenario, a Directory Service located in the Body Network layer fails (see Fig. 8). This failing Directory Service does not affect the rest of the Federation. However, the local set of stored Service Descriptions becomes unaccessible for any clients. The rest of the Descriptions stored in the other Directory Services are still available for the Clients, enabling them to continue working with a restricted number of Service Descriptions. The Federation still processes all five operations (search, register, deregister, modify and get Meta Data). There exists a mechanism allowing Directory Services to recover after a breakdown. This is explained in the recovery section below.

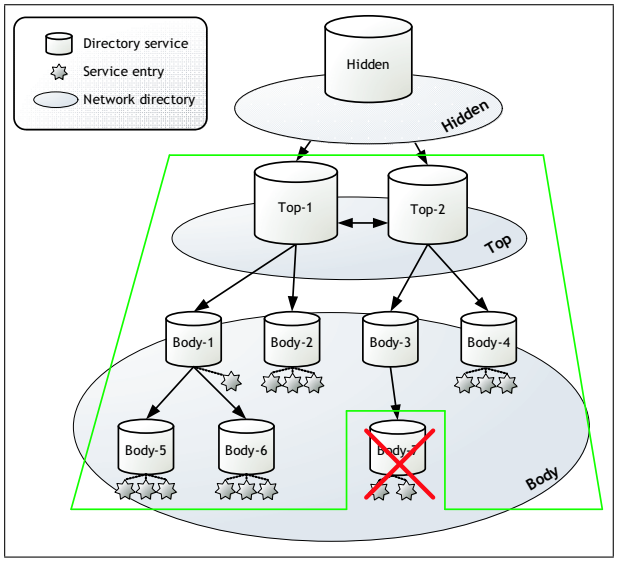

Figure 8: A WSDir Federation with one Directory Service from the Body Network layer is failing. The quadratic border defines the group of currently accessible Service Descriptions stored in the Federation.

In a second failure scenario, it is a Directory Service located in the Top Network layer that fails (see Fig. 9). The Federation is 'amputated' by the failing Directory Service's branch. In this case also, the rest of the Federation remains operational but all the Service Descriptions stored under the failing Directory Service become unavailable. Thus, several actions can be triggered while the Directory Service is down:

1. The clients (Discovery Agents) have a preconfigured list of addresses of Directory Services that are operating on the Top Layer Network. Thanks to this list, the clients can still access the Federation by picking up a new address from the list and simply contacting another Directory Service in the Top layer Network.

2. Directory Services in the Body layer Network that are operating under the failing Directory Services can also have a list of addresses of Directory Services in the Top layer Network. Being notified that the current Directory Service in which they registered fails, they can register themselves in another Directory Service operating in the Top layer Network. Most of the Service Descriptions stored in the Federation would then be accessible again.

These two mechanisms enable the clients to continue working with the Federation as well as providing the maximum number of Service Descriptions available to those Clients. The failing Directory Service can recover

using the recovering mechanism described in the following section.

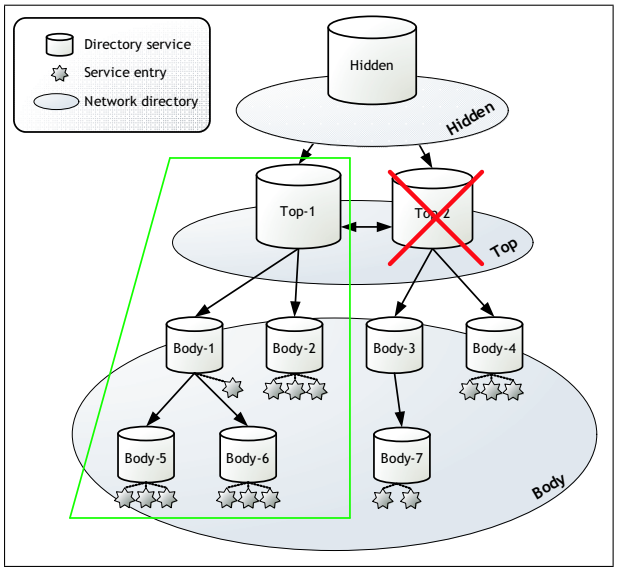

Figure 9: A WSDir Federation with one Directory Service from the Top Network layer is failing. The quadratic border defines the group of currently accessible Service Descriptions stored in the Federation.

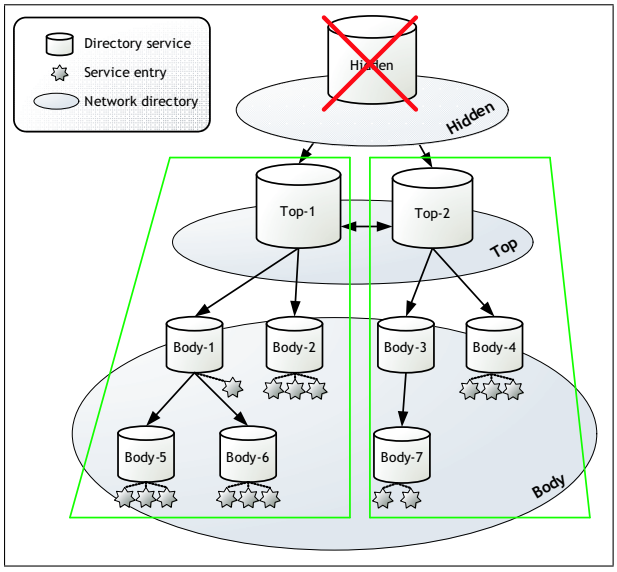

Figure 10: A WSDir Federation with the Directory Services from the hidden Network layer is failing. The quadratic borders define several groups of currently accessible Service Descriptions stored in the Federation.

In a third failure scenario, it is the Directory Service in the Hidden layer that breaks down (see Fig. 10). In this case, the Directory Services in the Top layer cannot communicate with each other anymore. This has the effect of creating groups of available Service Descriptions. A Client requesting a Directory Service from the top layer will only receive a restricted number of Service Description. Although the Federation is still operational, it would be better for Clients to access all Service Descriptions. Thus, to cope with that, Top Network layer's Directory Services use the same mechanism as those in the Body Network layer. They can be set with a list of addresses of Directory Services operating in the Hidden Network layer and registered in them when the regular one fails. In this case, several backup Directory Services must be ready. The failing Directory Service can recover using the recovering mechanism described in the following section.

10.2 Recovery

WSDir has been designed to cope with breakdowns by always keeping some parts of the Federation operational. When a Directory Service is down, the Service Descriptions stored in its memory are not accessible. Once the server works fine again, the Directory Service can be restarted[4]. When it restarts, it searches for a specific internal database that has been specified in the Directory Service's configuration file. Directory Services use this database to log all insert and modification requests coming from outside clients. Delete requests erase a specific entry in the database. It contains the following columns: i) the directory token of the Service Description; ii) the lease time during which the Service Description is supposed to stay stored in the Directory Service; iii) the full registration/modification request as sent by the Client.

The directory token is the primary key of the table. It is a unique identifier for each Service Description. The lease time is also stored to ensure that when a Directory Service restarts, the elapsed time of the failure is taken into consideration. Finally, the registration/modification request is stored in the database for the following reason: rather than creating a fixed and structured schema in the database to support a specific semantic language (such as OWLS), the full request is stored as a block and then decoded by the Directory Service during recovery. By doing so, WSDir can be easily adapted to store other types of Web service semantic languages.

10.3 Security

Regarding WSDir's vulnerability, there is an important issue about client authentication. For several applications, it is crucial to restrict access and interaction to trustful clients, avoiding requests from malicious entities. WSDir copes with this issue at two levels. First, clients have to provide in their requests both a sender and a receiver identification[5]. Furthermore, they may be

[4]The starting procedure is done by invoking a start method on the particular Directory Service

[5]This is part of the mandatory fields in the SL0 messages the Clients creates for each request

asked to provide an encrypted password and/or a K.509 certificate. The Directory Service serving the request can then decide if the client trustful or not. On the second level, WSDir uses policies, binding operations to directories. Depending on the policies, some of the authentication information will be used to grant or deny access to a specific clients. On the other hand, as WSDir runs as a Web service itself and uses the HTTP protocol, it may also take advantage of the HTTPS protocol. This mechanism ensures a client that it is contacting a trustful Directory Service.

11 PERFORMANCE RESULTS AND DISCUSSION

WSDir has been tested within the CASCOM project. During this testing process, WSDir proved to operate correctly all functions regarding requests. In the following sections, we discuss the quantitative performances of WSDir alone.

The objective is to compare the response time of a given WSDir Federation towards multiple simultaneous abstract[6] search requests sent by an increasing number of Clients[7].

We have decomposed the testing of WSDir into three scenarios. The difference between each scenario is the number of Directory Service units operating in the Federation.

For every single simulation, a set of Service Descriptions is stored[8] in the Federation and a given set of clients starts sending requests to the Federation, logging elapsed time of each request.

The Directory Services were partially distributed (some were running on the same servers) in a secured network. The Clients were all running at the same time in threads on the same computer. Each Client performed three random search requests to produce different results.

In CASCOM, the WSDir's Federation is composed of three network layers[9]: a Hidden Layer, a Top Layer and a Body Layer. Clients only access the Directory Services located in the Top Layer. Each scenario has a specific number of Directory Services in the Top Layer and the Body Layer (the Hidden Layer always contains one Directory Service). Our test scenarios are the following: i) *scenario 1* has three Directory Services in the Top Layer and six in the Body Layer; ii) *scenario 2* has six Directory Services in the Top Layer and six in the Body Layer; and iii) *scenario 3* has ten Directory Services in the Top Layer and twenty in the Body Layer.

Figures 11 to 13 show the average response time in milliseconds per number of stored services. Each line corresponds to a given number of simultaneous clients requesting the federation.

The important observation to make is that Scenario 2 (see Fig. 12) shows the best performance time. This implies that the Top Layer's Directory Services play an essential role in the scalability of the system. For a given number of Clients, we see that the time increases linearly. The abstract search algorithm is in O(n) complexity. The variations on the lines are due to run time performance clean up of the Directory Services and random queries (a query that matches a lot of Service Descriptions takes more time to be processed).

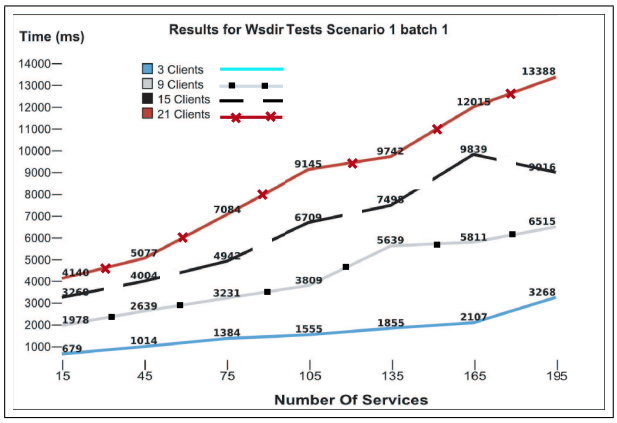

Figure 11: WSDir average search request processing time per number of services for scenario 1

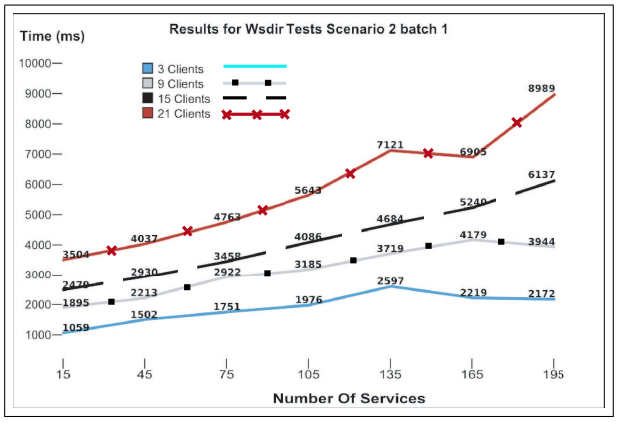

Figure 12: WSDir average search request processing time per number of services for scenario 2

[6]There are three types of search requests: abstract, grounding and matchmaker. The abstract and grounding types share the same complexity (O(n)). The matchmaker type is highly dependent on the matchmaker module used by WSDir (OWLS-MX for the CASCOM project).

[7]The Clients are Threads that create appropriate SOAP messages, send them to a predefined target address and log the response time in millisecond. In CASCOM, those clients are Discovery Agents

[8]These Service Descriptions are evenly distributed over the Directory Service units of the Federation.

[9]Network layers are used to model the network topology

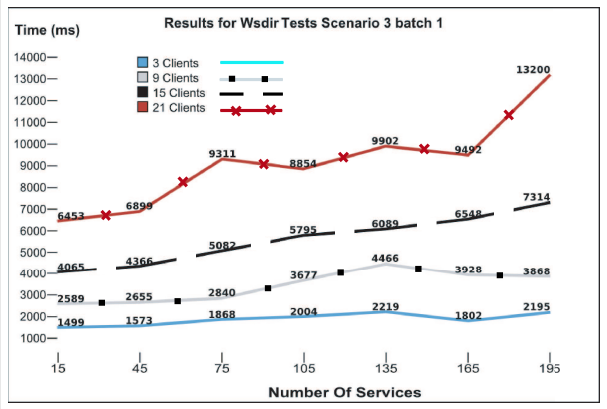

Figure 13: WSDir average search request processing time per number of services for scenario 3

12 RELATED WORK

In this section, we discuss related research in semantic web services discovery. We are considering only systems that have been fully implemented, as it is the case for WSDir.

The METEOR-S discovery framework [9] copes with the problem of discovering services in a scenario where service providers and requesters may use terms from different ontologies. Based on user's ontology, METEOR-S's Web Service Discovery Infrastructure organizes multiple registries[10], enabling semantic classification of all Web services based on domains. Their approach relies on annotating semantically service registries (for a particular domain) and exploiting such annotations during discovery. This system can be deployed in a Peer-to-Peer network, relying on the JXTA[11] project, making it scalable. Two algorithms have been implemented. One for semantic publication of Web services and the second one for discovery of those Web services.

This project differs from WSDir by allowing multiple ontologies to be used at the same time. This approach solves an interoperability issue that WSDir doesn't treat, although WSDir has an open architecture for any types of ontologies. In contrast to METEOR-S which implements a dedicated algorithm for discovery, WSDir can use many matchmaking modules for discovery. This thus allows WSDir to be more flexible for specific use cases.

GLUE [3] is a WSMO[12] compliant discovery engine that aims at developing an efficient system for the management of semantically described Web Services and their discovery. GLUE is built around an open source f-logic inference engine called Flora-2[13] that runs over XSB[14]. The basis of the GLUE infrastructure is a set of facilities for registering and looking up WSMO components (ontologies, goals, Web Service descriptions and mediators). With the use of these components, GLUE implements a matching mechanism that relies on wgMediators. Requester entities register a class of goals. Discovery is then performed by submitting goals. Similarly, providers register first a class of Web service descriptions and then publish Web service descriptions. The link between a class of Web services and a class of goals is embedded in a dedicated wgMediator, that uses a set of f-logic rules to assert similarities. In contrast to the GLUE approach where a central storage unit is used with a single inference engine, WSDir avoids bottle neck problems by distributing its Directory Services; therefore, all requests are splitted between several Directory Services.

The WSPDS system [1] is also a peer-to-peer discovery system that is enabled with semantic matchmaking. In WSPDS, WSDL files need to be semantically annotated in order to be available for discovery. This is done by using the WSDL-S framework[15]. By doing so, the WSDL-S file doesn't have to know anything about the ontology being used by a Web service description file such as OWL-S or WSMO. The system is built around a peer-to-peer architecture, where peers act as servants (acting both as clients and servers). Discovery queries can be sent to any servant, that will forward the query to its neighbors. All the communication is done via SOAP messages. WSDir aligns itself very closely to the WSPDS service. They use both a Web service interface and they rely on a peer-to-peer architecture. The main difference is in the work to be done for new ontologies. In WSPDS, each WSDL file needs to be re-written using the WSDL-S framework. In contrast, WSDir simply needs to add the appropriate matchmaking module.

[7] describes a framework for semantic Web service discovery. This framework is based on context specific mappings from a user ontology to a specific domain ontology. Using these mappings, the user queries are then transformed into a specific form of query. These queries can be processed by a match making engine that takes in consideration the domain ontologies and the stored Web services. In the prototype implementation, the match

[10] Web service registries for publishing Web services
[11] See https://jxta.dev.java.net/
[12] WSMO - Web Service Modeling Ontology, see http://www.wsmo.org/
[13] See http://flora.sourceforge.net/
[14] XSB is an open source implementation of tabled-prolog and deductive database system. See http://xsb.sourceforge.net/
[15] see http://www.w3.org/Submission/WSDL-S/

making engine is based on JESS and JENA, that uses a JESS knowledge base. When service providers store their services, the Service Registery API parses and converts the OWL ontology into a collection of JESS facts, and stores them in a knowledge base. This project unifies multiple ontologies and copes with interoperability issues, making them transparent for the user. But like the GLUE project, it has a bottle neck architecture because of its central storing unit. In contrast, WSDir uses a distributed architecture.

13 CONCLUSION

WSDir has been tested thoroughly in a real distributed setting spread over different countries. The system has proven to be scalable and very stable. We have integrated the system in a use case scenario in the pervasive eHealth domain that uses WSDir as its backbone. Among others, future work could enhance the following aspects.

From the security and privacy-awareness point of view, we currently employ standard security mechanisms for accessing the directory services. In particular, if a directory service requires protecting messaging from overhearing or if it would require privacy sensible data as parameters, the access to this web service will be based on HTTPS. In cases where no HTTPS is available, we could couple WSDir with Guarantor agents [2] spread in the architecture in order to provide a secure tunneling between agent messages and HTTPS.

Another improvement could define security measures directly within the directory system by defining specific policies. A policy can be employed to restrict the right to perform a certain operation on a directory to only those clients that can provide the right credentials. Using this method, registration of services to a directory and search operations on directories can be restricted. For example, a directory service that does not forward any queries pertaining to a ”Hospital” domain directory will simply return its entries for the domain and nothing more. This would be completely transparent to the requestor, as its view of the network topology is determined by the application of policies of the directory services underneath it.

As mentioned in the usability section 9, administrating and monitoring WSDir was not a major priority during its development. Although several tools have been developed to cope with testing issues, the system lacks consistency. Some of these tools should be enhanced and packaged into a single administration package. Beyond that, a complete WSDir editor should be developed to help administrators setting up easily networks of Directory Services.

References

[1] F. Banaei-Kashani, C.-C. Chen, and C. Shahabi. Wspds: Web services peer-to-peer discovery service. In *Proceedings of the International Symposium on Web Services and Applications(ISWS'04)*, Nevada, June 2004.

[2] R. Bianchi, A. Fontana, and F. Bergenti. A real-world approach to secure and trusted negotiation in mass. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 1163–1164, New York, NY, USA, 2005. ACM Press.

[3] E. Della Valle, D. Cerizza, and I. Celino. The mediators centric approach to automatic web service discovery of glue. In *Proceedings of the First International Workshop on Mediation in Semantic Web Services: MEDIATE 2005*, Amsterdam, Netherlands, December 2005.

[4] Foundation for Intelligent Physical Agents. Fipa sl content language specification, December 2002.

[5] M. Klusch, B. Fries, and M. Khalid. Owls-mx: Hybrid semantic web service retrieval. In *Proceedings 1st Intl. AAAI Fall Symposium on Agents and the Semantic Web*, Arlington VA, USA, 2005.

[6] D. Martin, M. Paolucci, S. McIlraith, M. Burstein, D. McDermott, D. McGuinness, B. Parsia, T. Payne, M. Sabou, M. Solanki, N. Srinivasan, and K. Sycara. Bringing semantics to web services: The owl-s approach. In *Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, 2004.

[7] J. Pathak, N. Koul, D. Caragea, and H. V. A framework for semantic web services discovery. In ACM, editor, *Proceedings of the ACM 7th Intl. workshop on Web Information and Data Management (WIDM-2005)*, 2005.

[8] M. Schumacher, T. van Pelt, I. Constantinescu, A. de Oliveira e Sousa, and B. Faltings. Wsdir: a federated directory system of semantic web services. In IEEE, editor, *Proceedings of the16th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE 2007)*, 2007.

[9] 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*, 2004.