Chapter 9

Distributed Directories of Web Services

**Michael Schumacher, Alexandre de Oliveira e Sousa,
Ion Constantinescu, Tim van Pelt and Boi Faltings**

9.1 Introduction

This chapter presents WSDir, the federated directory system used in CASCOM. Its main functionality is to let heterogeneous Semantic Web Service descriptions be registered and searched by certain clients. As such, it realizes a lookup function with basic retrieval schemes.

There are several main requirements for a distributed directory system. First, it should be easy to invoke by any client. This led us to define a *Web Service interface* to WSDir: it is a universally accepted standard, it provides a well-defined method to use the directory, and it allows for interacting with a heterogeneous set of clients. The sole requirement on the part of the client is that it should be able to communicate over a Web Service interface. Second, the nature of the applications to be realized requires the directory system to be distributed, for instance applying a geographical specialization of the directories. Third, 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. Fourth, the directory should allow a great number of services to be registered, and this in a very dynamic way, including lease times.

Our 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.

The chapter is organized as follows. In Section 9.2, we explain the service

entries of Semantic Web Services that can be stored in WSDir. Sections 9.3 to 9.6 explain the architecture of WSDir by presenting *directories*, *directory services*, *directory operations*, and *policies*. In Section 9.7, we give the concrete network architecture used in CASCOM. Sections 9.8 and 9.9 discuss respectively usability and vulnerability issues of WSDir. After referring related work in Section 9.10, we conclude the chapter in Section 9.11.

9.2 Service Entries

Services are described using the Web Ontology Language for Web Services, OWL-S [4]. Internally, the directory system stores then service entries in the FIPA SL0 description language. SL0 has been chose because the whole CASCOM infrastructure is using this language for interpretability between agents. Furthermore, as we achieve with SL0 an independent way to store any kind of services and not only OWL-S descriptions.

The internal service representation contains a subset of the information provided in the original service 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.

In the following, we present the information that a service entry in WSDir contains:

ServiceCategories Refers to an entry in some ontology or taxonomy of services. The value of the property is a set containing elements of the class *ServiceCategory*, which is defined in the OWL-S ontology. The information is ultimately derived from the service categories defined in the service profile.

ServiceProfileURIs This slot contains a set of profile URIs that is referred to in the service description. If the profiles are included in the service description as full-text, no URIs are stored. The URIs point to an externally stored, but (web-)retrievable service profile.

ServiceProcessURI A process URI that is defined in the service description. If none is included for the service description, the slot will be empty. If the process is included in the service description as full-text, no URI is stored. The process URI points to an externally stored, retrievable service process.

ServiceGroundings The slot that contains a set of full-text service groundings for the service. Empty set if no grounding is associated with the service (abstract service). Makes it possible to retrieve only service groundings.

OWLSServiceDescription The slot that contains the original OWL-S service description as a full-text entry. Service profile(s) and process may be referred to as URIs, though service groundings must be included as full-text.

9.3 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 organisational structure (peer-to-peer, hierarchy etc.). Specifically:

- A directory can contain other directories.
- A directory supported by one or more directory services.

The above is used to characterize directories by two different types of interactions:

1. *Client-Directory* interactions: in which clients 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.
2. *Director-Directory* interactions: in which 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 Section 9.7.1 on network topology.

9.4 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. It 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 9.1 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.

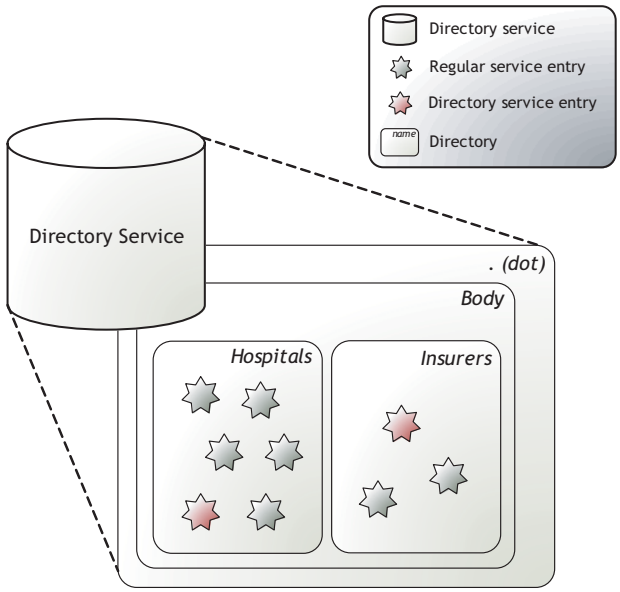

Figure 9.1: Visual recapitulation of directory system concepts

9.5 Directory Operations

The directory is able to handle five types of operations, corresponding to data manipulation primitives (*register, deregister, modify, search*) and the retrieval of meta-data information (*get-profile*). The methods are accessible remotely through a Web Service interface.

We present the operations the directory service interface offers. Before that, some of the objects that are passed as parameters are defined:

Identity-info The *identity-info* element is to be used for providing identification information regarding the invoker of a given operation in the form of an *actor-identifier*. This object can contain also a structured object providing credentials information that can be relevant to some form of authentication (e.g., a password, a X.509 certificate or a more sophisticated session key).

Directory-id The *directory-id* is a string that uniquely identifies a directory in the frame of a given directory service. It can include the names of other directories if the directory service uses a path like scheme for identifying nested directories.

Directory-token A directory-token refers to an entry registered within the directory and can be used to identify the registration entry. A directory-token contains a key (string) that uniquely identifies a particular entry in a given

directory. The directory service usually generates these keys but some directories may support the registration of entries under keys specified by the client of the directory service.

Structured-object *structured-object(s)* are the data elements stored in the directory which are subsequently available for search. The only assumption about structured objects is that they have a frame-like structure similar to RDF, the SOAP XML encoding or FIPA SL0. In principle, there are no restrictions on the content of the data of the structured objects. For our experiments, the *structured-objects* are encoded in FIPA SL0.

Hereafter, we describe all operations on WSDir (except *get-profile*):

Registration

register (identity-info, directory-id, directory-token, structured-objects, lease-time)

The *register* operation enables a client to register a service description entry (as specified in the *structured-objects*) into a *directory* for a time period given by the *lease-time* parameter. If a directory-token is specified when the operation is requested then the directory should try to register the new entry using the key specified in the *directory-token*. Upon successful registration the directory service returns an *ok* message containing either a new *directory-token* or the original *directory-token* if it was specified by the user together with a new *lease-time*.

The *directory-service* can also return a *redirect* message pointing to another directory where the client could try to register its object. The detailed semantics of the *redirect* message are dependent on the policies governing the directory for which the redirect was issued. For example, this may happen for load-balancing reasons in top-level directories. To uphold the transparency of the federated nature of the system to the client, the redirection will be opaque. Note that a register policy of the directory may also forward the registration request to another directory, e.g., to balance the load of the directory.

Deregistration

deregister (identity-info, directory-id, directory-token)

The deregister operation de-registers a service that previously has been registered identified. The entry to be de-registered is included in the directory-token that has been obtained at registration.

Modification

modify (identity-info, directory-id, directory-token, structured-objects, lease-time)

The modify operation allows modifying a registered service. The structured object itself can be modified or the parameters of the entry, such as the lease-time.

Search

search (identity-info, directory-id, structured-objects, search-constraints)

The *search* operation looks in the directory for services that match a template (*structured-objects*). 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:

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

In Section 9.7.4, we show an example of how a search procedure is internally handled.

9.6 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 Policies of this type 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.

Reactive policies These 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

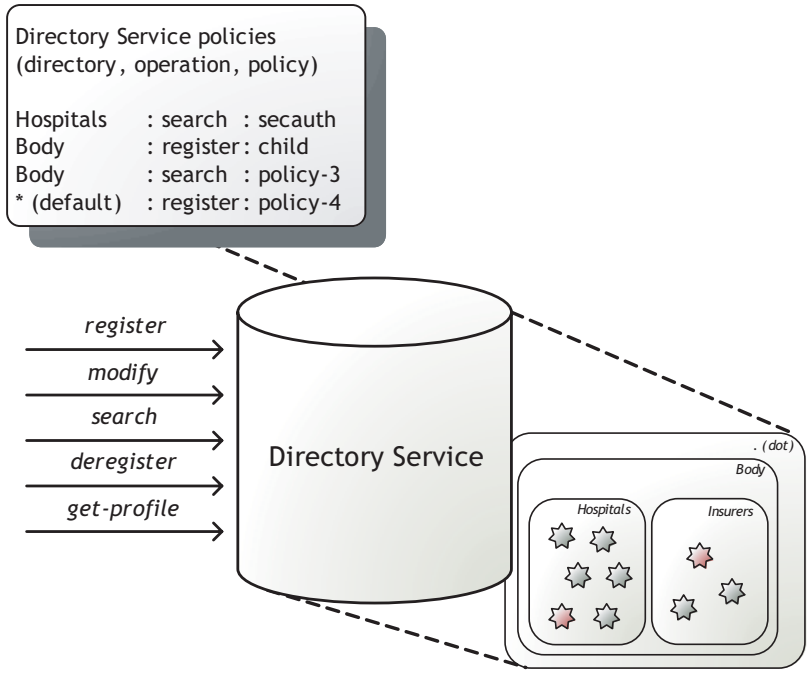

Figure 9.2: Example use of policies

will be accepted. Each directory service can define its own policies or use one of the pre-defined 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 9.2 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.

9.7 CASCOM Service Directory 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 is the network architecture that has been used in the CASCOM infrastructure.

9.7.1 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;
- Forwarding search only: since there is no replication, directory services will forward queries to other directory services;
- Query message duplicate checking: 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;
- No results duplicate checking: identical 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.

In the network, 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 Figure 9.3.

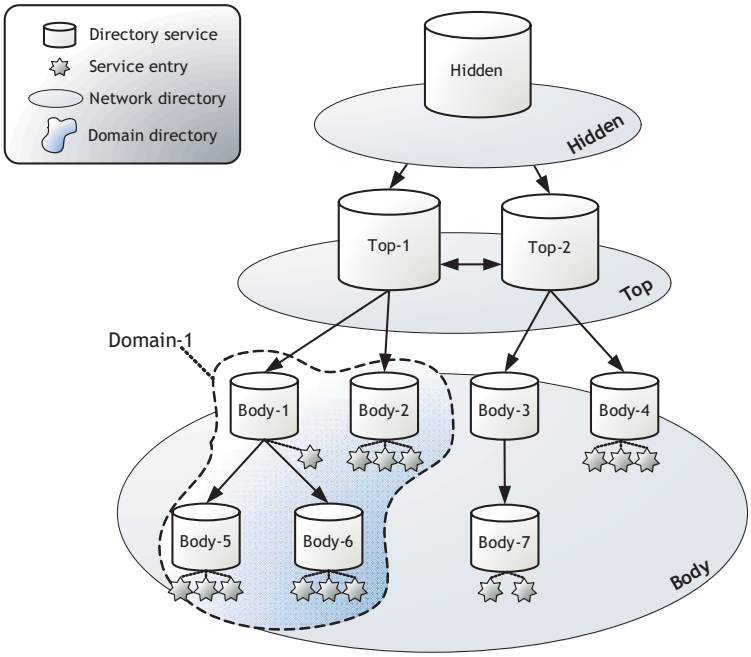

Figure 9.3: Network topology

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 identity-info 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.

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.

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.

9.7.2 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. Instead of an OWL-S Profile, the passed structured object contains a *directory-service-profile* object.

9.7.3 Used Directory Policies

WSDir employs a set of pre-defined pro-active policies, mainly for routing purposes. Figure 9.4 gives an overview of the pre-defined policies and their hierarchy. We do note present here the details of each policy. However, we show in Figure 9.5 how these policies are applied to construct the basic network topology . Per network directory, the set of policies in place is listed.

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.

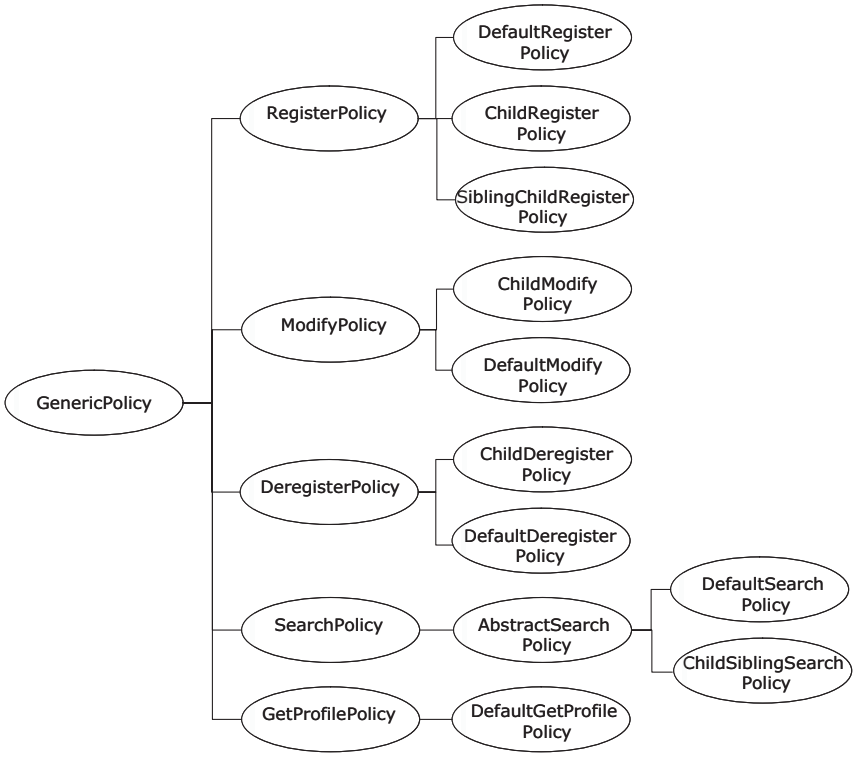

Figure 9.4: Predefined policy tree

9.7.4 Examples of Network Interactions

The following section describes two examples 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.

We illustrate the process of query resolution in Figure 9.6 and 9.7: a client issues a search request for service profiles matching a template in the *Hospital* domain directory. The template is provided in the form of a *structured-object*.

1. 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.
2. The *Top-2* directory service forwards the query to its siblings.
3. Directory services that have an entry for the *Hospital* domain directory in their service profile propagate the query down.

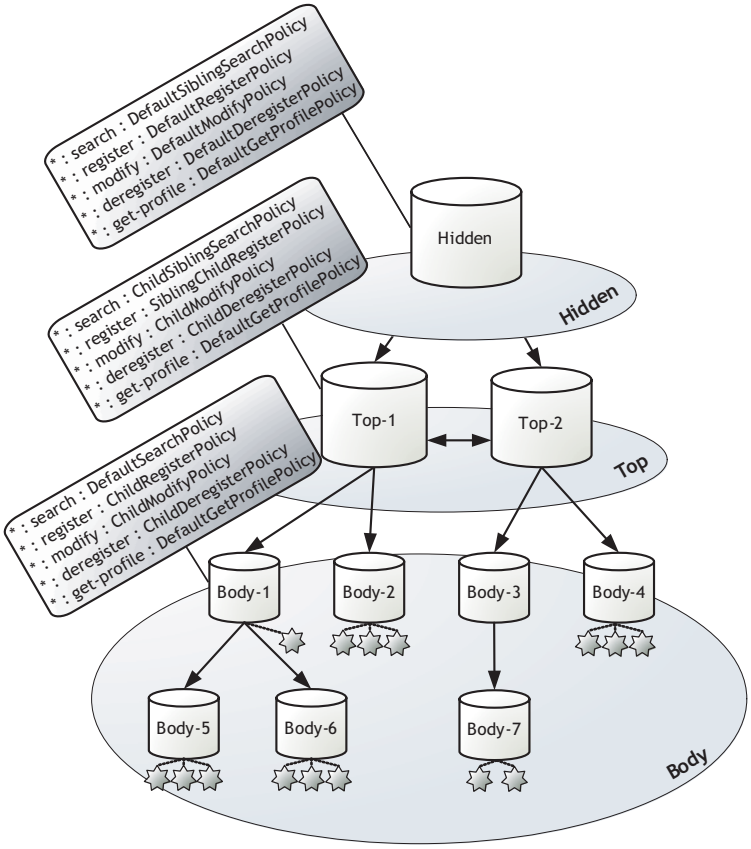

Figure 9.5: Policies in the network topology

4. To guarantee full query resolution, the query is forwarded to all directory services that are known to store entries for the *Hospital* domain directory.

5. 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.

The next sections will now discuss the usability, vulnerability and performance of WSDir.

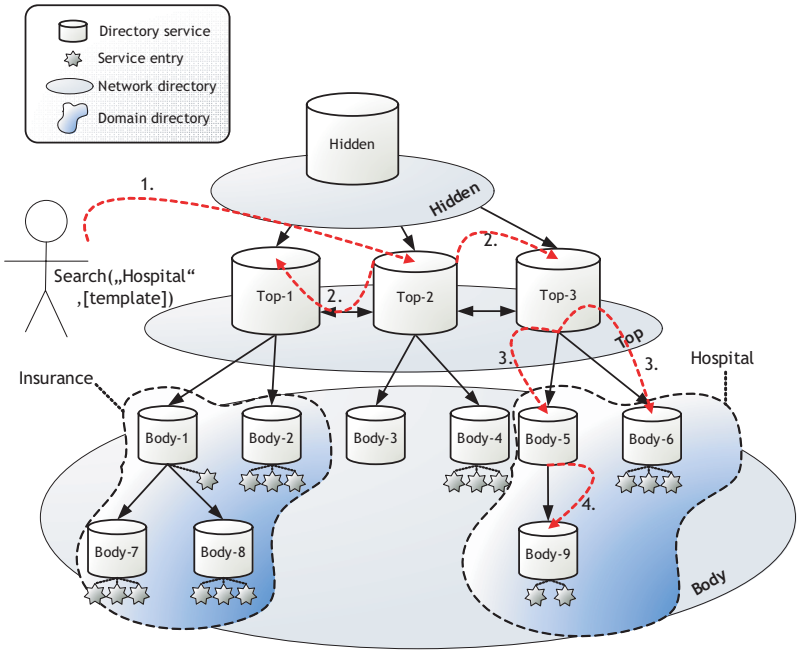

Figure 9.6: Query resolution (1)

9.8 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 necessary 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

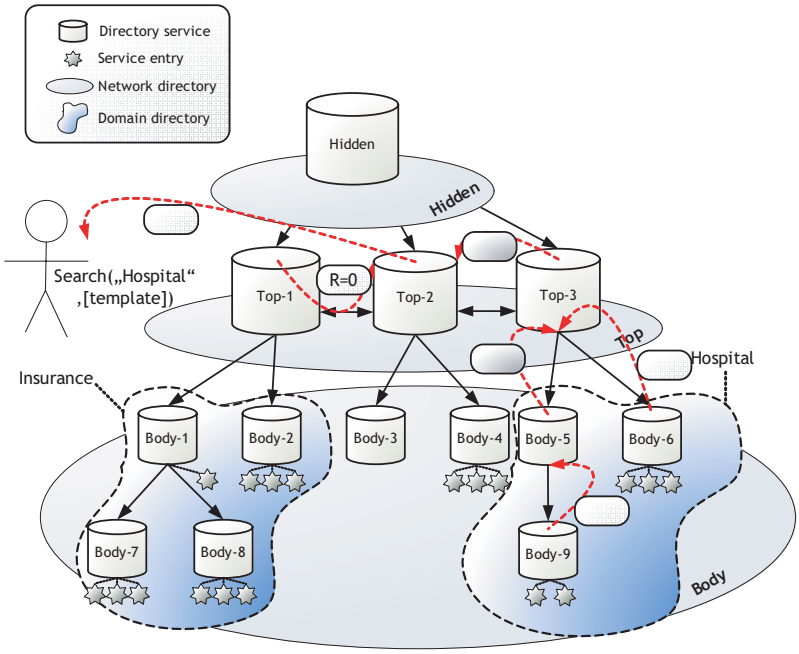

Figure 9.7: Query resolution (2)

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.

9.9 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.

9.9.1 Breakdowns

In the CASCOM project, we have used the network topology presented in 9.7.1, 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 9.8 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).

In a first failure scenario, a Directory Service located in the Body Network layer fails (see Figure 9.9). 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.

In a second failure scenario, it is a Directory Service located in the Top Network layer that fails (see Figure 9.10). 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 pre-configured 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 Ser-

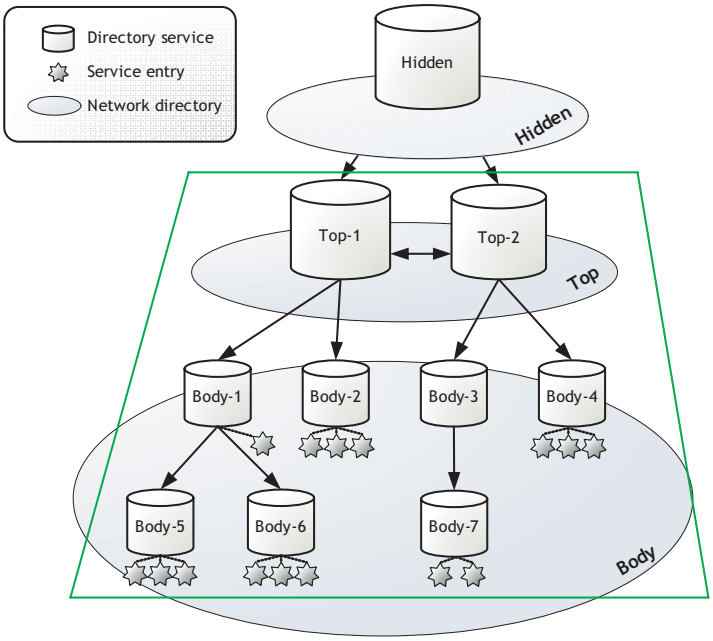

Figure 9.8: A WSDir Federation with all the Directory Services from each Network layer is working correctly. The quadratic border defines the group of currently accessible Service Descriptions stored in the Federation.

vices 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.

In a third failure scenario, it is the Directory Service in the Hidden layer that breaks down (see Figure 9.11). 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

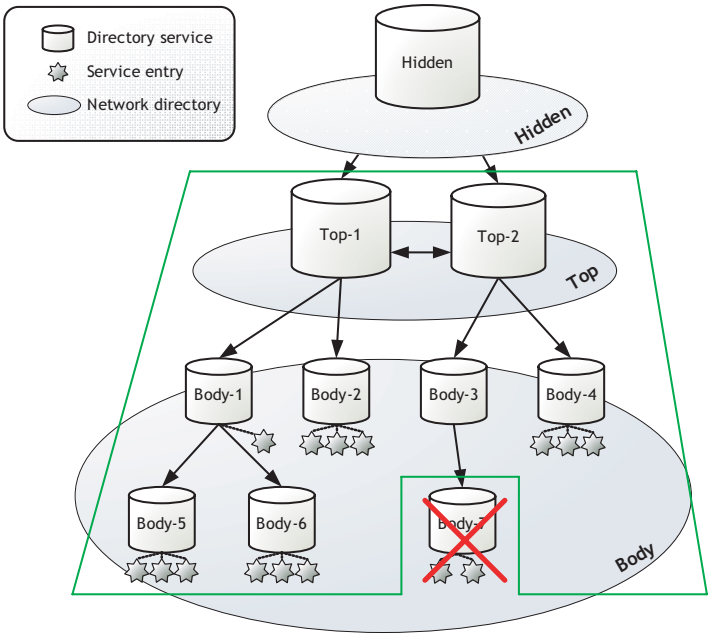

Figure 9.9: A WSDir Federation with one Directory Service from the Body Network layer is failing. The quadratic border defines the group of currently accessible Service Descriptions stored in the Federation.

Directory Services must be ready. The failing Directory Service can recover using the recovering mechanism described in the following section.

9.9.2 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[1]. 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

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

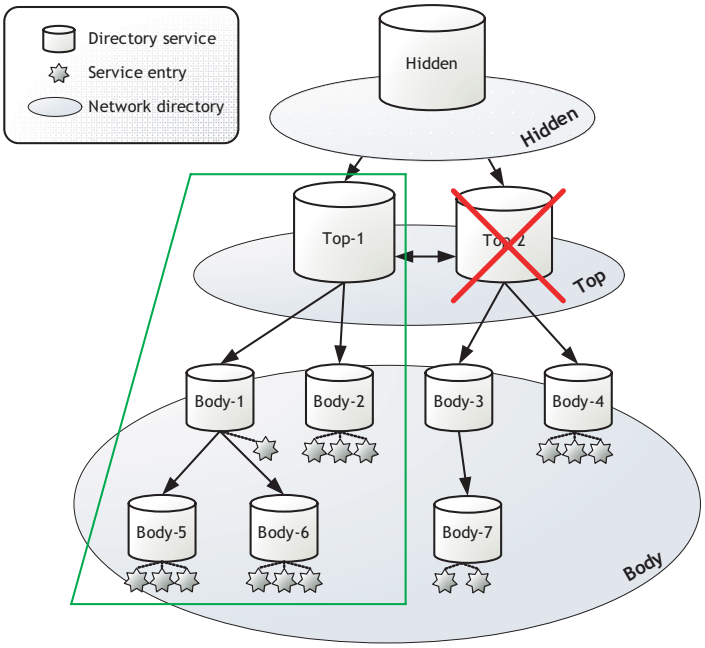

Figure 9.10: A WSDir Federation with one Directory Service from the Top Network layer is failing. The quadratic border defines the group of currently accessible Service Descriptions stored in the Federation.

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.

9.9.3 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[2]. Furthermore, they may be asked to provide an encrypted password and/or a K.509 certificate. The Directory Service serving the

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

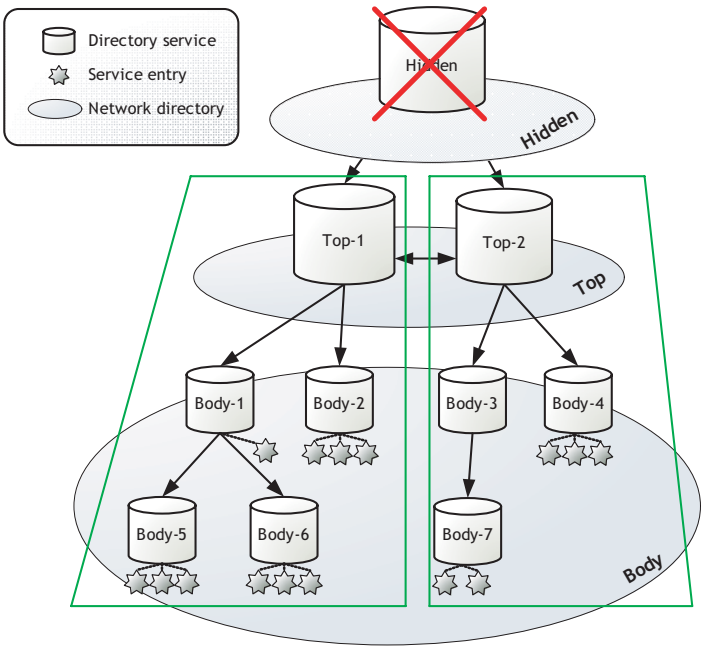

Figure 9.11: A WSDir Federation with the Directory Services from the hidden Network layer is failing. The quadratic borders define several groups of currently accessible Service Descriptions stored in the Federation.

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.

9.10 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 [6] 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[3], enabling semantic classifi-

[3]Web Service registries for publishing Web Services

cation 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[4] 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[5] 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[6] that runs over XSB[7]. 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[8]. 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

[4]See https://jxta.dev.java.net/

[5]WSMO - Web Service Modeling Ontology, see http://www.wsmo.org/

[6]See http://flora.sourceforge.net/

[7]XSB is an open source implementation of tabled-prolog and deductive database system. See http://xsb.sourceforge.net/

[8]see http://www.w3.org/Submission/WSDL-S/

WSDL file needs to be re-written using the WSDL-S framework. In contrast, WSDir simply needs to add the appropriate matchmaking module.

[5] 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 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.

9.11 Summary

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 the CASCOM use case scenario. 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 9.8, 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 administra-

tion 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] 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.

[5] J. Pathak, N. Koul, D. Caragea, and Honavar 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.

[6] 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.