A Distributed Approach for the Federation of Heterogeneous Registries

Luciano Baresi and Matteo Miraz

Politecnico di Milano,
Dipartimento di Elettronica e Informazione, Milano, Italy
{baresi, miraz}@elet.polimi.it

Abstract. Registries play a key role in service-oriented applications. Originally, they were neutral players between service providers and clients. The UDDI Business Registry (UBR) was meant to foster these concepts and provide a common reference for companies interested in Web services. The more Web services were used, the more companies started create their own "local" registries: more efficient discovery processes, better control over the quality of published information, and also more sophisticated publication policies motivated the creation of private repositories.

The number and heterogeneity of the different registries —besides the decision to close the UBR— are pushing for new and sophisticated means to make different registries cooperate. This paper proposes *DIRE* (DIstributed REgistry), a novel approach based on a publish and subscribe (P/S) infrastructure to federate different heterogeneous registries and make them exchange information about published services. The paper discusses the main motivations for the P/S-based infrastructure, proposes an integrated service model, introduces the main components of the framework, and exemplifies them on a simple case study.

1 Introduction

Service-oriented applications exploit *registries* to expose services to possible clients. Originally, the registry was a *neutral* actor between clients and providers; it was a "shared" resource aimed at facilitating their cooperation. This was the original mission of the first version of the UDDI (Universal Description, Discovery, and Integration, [11]) specification, which was the first market-supported standard that allowed companies to publish their services and interact with clients [2]. To this end, in September 2000, BEA, IBM, and Microsoft started *UBR* (UDDI Business Registry), a public UDDI-based registry, but also a common and neutral reference for all the companies interested in publishing and exploiting Web services.

The diffusion of Web services led to the need for "private" registries, directly controlled by the different companies, in parallel with the public one. Companies wanted to be able to control their registries to increase the efficiency of the discovery process, but they also wanted to manage private information —e.g., exclusive offers for their clients. These registries did not substitute the central

A. Dan and W. Lamersdorf (Eds.): ICSOC 2006, LNCS 4294, pp. 240–251, 2006.

one, which continued to be a universally-known reference. If a company was not able to serve a request internally, it could always access the central repository to find the services it needed.

Both the second version of the UDDI specification [11], and other approaches, like ebXML [5], took a more decentralized view and allowed for the creation of different separated registries. Moreover, January this year, the companies behind UBR decided to shut it down [6] since the original goal —the creation of a common sandbox to foster the diffusion of the service-oriented paradigm— was met. The new advice is to install a dedicated repository for each company. The complete control over published information allows the company to select and filter the information it wants to publish, organize it the way it prefers, and thus better tune the discovery process. Usually, companies manage their services, and those provided by their partners, but the lack of a common search space hinders the discovery of new services —supplied by providers with which the company is not used to cooperate. Companies interested in new services must a-priori select the companies that might provide them, and then search their proprietary registries, if allowed. Moreover, clients do not often know the services that fit their needs directly, but they would like to query the registries to find those that better fit their expectations. The more accurate the descriptions associated with services are, the more precise the discovery can be.

To overcome the lack of a centralized repository, and also to supply an extensible model to describe services, this paper proposes *DIRE* (DIstributed REgistry), a novel approach for the seamless cooperation among registries. *DIRE* proposes a decoupled approach based on the publish and subscribe (P/S) middleware *ReDS* [3]. A unique distributed dispatcher supports the information exchange among the different registries. Even if the dispatcher is logically unique, it provides a physically distributed communication bus to allow registries to publish information about their services, and clients —which may be other registries or suitable application interfaces— to receive it. According to the P/S paradigm, clients must subscribe to the services they are interested in and then the dispatcher forwards relevant data as soon as published. Moreover, *DIRE* supports the creation of dedicated *federations* to allow for the re-distribution of interesting information among the members even if they are not subscribed to it directly.

DIRE fosters the integration of repositories implemented using different technologies (currently, UDDI and ebXML) by means of dedicated plugs called *delivery managers*. They adopt a unique service model that both extends and abstracts the models used by the single registries, and provides a flexible means for the characterization of services. Delivery managers are also in charge of the interaction of registries with the P/S communication bus.

The rest of the paper is organized as follow. Section 2 surveys similar proposals and paves the ground to our approach. Section 3 sketches the approach. Section 4 describes the technology-agnostic service model defined to document services and provide delivery managers with a common base. Section 5 presents the delivery manager and digs down into the mechanisms offered to support flexible

cooperations among registries. Section 6 demonstrates the approach on a simple case study and Section 7 concludes the paper.

2 Related Work

The different approaches for the cooperation and coordination among registries can be roughly classified in two main groups: approaches based on *selective replication* and approaches based on *semantic annotations*.

UDDI and ebXML belong to the first family. UDDI v.3 [12] extends the replication and distribution mechanisms offered by the previous versions to support complex and hierarchical topologies of registries, identify services by means of a unique key over different registries, and guarantee the integrity and authenticity of published data by means of digital signatures. The standard only says that different registries can interoperate, but the actual interaction policies must be defined by the developers.

ebXML [5] is a family of standards based on XML to provide an infrastructure to ease the online exchange of commercial information. Differently from UDDI, ebXML allows for the creation of federations among registries to foster the cooperation among them. The idea is to group registries that share the same commercial interests or are located in the same domain. A federation can then be seen as a single logical entity: all the elements are replicated on the different registries to shorten the time to discover a service and improve the fault tolerance of the whole federation. Moreover, registries can cooperate by establishing bilateral agreements to allow registries to access data in other registries.

Even if these approaches foster the cooperation among registries, they imply that all registries comply with a single standard and the cooperation needs a set up phase to manually define the information contributed by each registry. Moreover, given the way relations are managed by UDDI [8], the more registries we consider, the more complex the management of relations become, and the cost of publishing or discovering a service increases.

METEOR-S [13] and PYRAMID-S [7] are the two main representatives of the second family. They support the creation of scalable peer-to-peer infrastructures for the publication and discovery of services. METEOR-S only supports UDDI registers, while PYRAMID-S supports both UDDI and ebXML registries. Both the approaches adopt ontology-based meta-information to allow a set of registries to be federated: each registry is "specialized" according to one or more categories it is associated with. This means that the publication of a new service requires the meta-information needed to categorize the service within the ontology. This information can be specified manually or it can be inferred semi-automatically by analyzing an annotated version of the WSDL interface of the service. Notice that, if the same node of the ontology is associated with more than one registry, the publication of the services that "belong" to that node must be replicated on all the registries. Services are discovered by means of semantic templates. They give an abstract characterization of the service and are used to query the ontology and identify the registries that contain significant information.

The semantic infrastructure allows for the implementation of different algorithms for the publication and discovery of services, but it also forbids the complete control over the registries. Even if each registry can also be used as a stand-alone component, the selection of the registries that have to contain the description of a service comes from the semantic affinity between the service and the federated registries. For this reason, each node in a METEOR-S or PYRAMID-S federation must accept the publication of a service from any other member of the community. These approaches are a valid solution to the problem of federating registries, but the semantic layer imposes too heavy constraints on publication policies and also on the way federations can evolve dynamically.

3 DIRE

This section introduces the main elements of *DIRE* (DIstributed REgistry), which is our proposal to support loose interactions among registries. DIRE aims at fostering the communication among different proprietary registries by means of two elements: a distributed *communication bus* and a *delivery manager* associated with each registry. The former is introduced to interconnect the delivery managers, and is based on *ReDS* [3], a distributed publish and subscribe middleware. The latter acts as *facade*: it is the intermediary between the registry and the bus and manages the information flow in the two directions.

In P/S systems, components do not interact directly, but rather the communication is mediated by a *dispatcher*. Components send (*publish*) events (*messages*) to the dispatcher and decide the events they want to listen to (*subscribe*/*unsubscribe*). The dispatcher forwards (*notifies*) received events to all registered components.

DIRE allows registries to use the communication bus in two different ways:

- Registries (i.e., the companies behind them) declare their interest for particular services. This means that each publication —on a particular registry connected to the bus— is propagated to those registries that subscribed to it. Once the information is received, they can either discard it, or store it locally. The goal is to disseminate the information about services based on interests and requests, instead of according to predefined semantic similarities. This way, *DIRE* supports a two-phase discovery process. The registry retrieves the services the organization is interested in from the P/S infrastructure. The client always searches for the services it needs locally.
- Registries can be grouped in *federations* (communication groups, according to the P/S jargon) to allow for the re-distribution of interesting information among the elements of a federation even if they are not directly subscribed to it. To create a federation, some registries must declare their interest for a common *topic*. This means that when one of these registries is notified of new services, it re-distributes these data within the federation. Topics are not service types, but are abstract concepts, used to identify, for example, geographical areas, business organizations, or the parties involved in virtual enterprises. One registry can belong to different federations.

4 Service Model

The heterogeneity of considered registries and the need for a flexible means to describe services are the main motivations behind the *DIRE* service model. This model is conceived with three main goals: the compatibility with well-known standards, the unique identification of exchanged data, and the authentication of retrieved information. Currently, *DIRE* supports both UDDI and ebXML registries, but since we adopt JAXR (Java API for XML Registries, [9]), it is possible to easily integrate other types of registries. Our proposal (Figure 1) is in line with those approaches that tend to unify existing models (e.g., JAXR): we reuse the business elements, which are similar, and we introduce the concept of *facet* to allow for a more detailed and flexible management of the technical information associated with services.

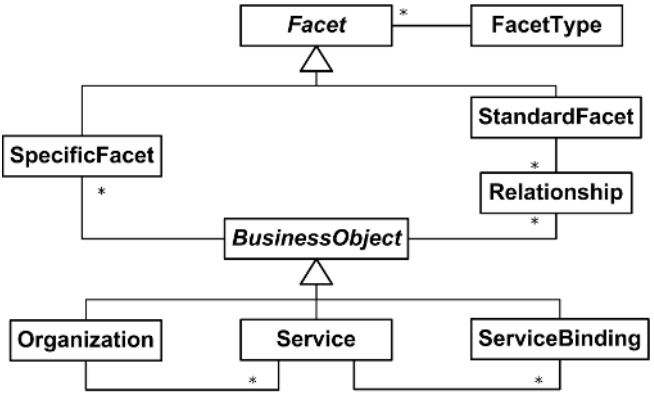

Fig. 1. *DIRE* information model

As far as business data are concerned, the lower part of Figure 1 describes the "shared" structure in terms of Organizations, Services, and ServiceBindings, with the meaning that these elements assume in existing registries.

The similarities vanish when we consider technical data. The different registries tend to provide predefined schemas to organize this information. In some cases, they also distinguish between references (stored in registries) and actual contents (put in repositories). *DIRE* integrates these elements and describes BusinessObjects by means of Facets. Each Facet addresses a particular feature of the BusinessObject by using an XML language. Facets are typed (FacetType) —based on the information they embed or the language they use— to ease the retrieval of services. For example, we can create *WSDL* Facets to describe the interfaces of a service, *RDF* Facets to add semantics, or *XMI* Facets to specify complex service behaviors through UML diagrams.

StandardFacets characterize recurring features (and are associated with BusinessObjects by means of Relationships). For example, they can specify the compliance

with an abstract interface or the quality level according to a given certification authority. Given their meaning, we assume that StandardFacets are managed by special-purpose authorities that are in charge of linking the different services to these "certified" characteristics. StandardFacets are also a way to define the compatibility level between two services [11]. SpecificFacets describe what is peculiar to a given service. It can be the policies that must be used to charge users or the testing data that the provider wants to share with its possible clients. Some aspects require a combination of these two types: for the WSDL interfaces, we suggest to adopt the proposal presented in [1,4] and distinguish between the standard interface of a service and the properties of a given implementation: a StandardFacet describes the former and SpecificFacets deal with the latter.

Every user can attach a facet to a BusinessObject, even if it is not the service provider. This feature lets each company use its local registry as a blackboard, and allows a decoupled communication among the different elements of the service-centric system (e.g., runtime monitors might create facets that are then used by the dynamic binder). *DIRE* allows providers to sign and share these facets, thus allowing the receivers —if they trust the provider— to get a more detailed knowledge of the services in their registry.

The distributed setting behind *DIRE* requires that identification and authentication be carefully addressed. Since we can hardly understand the source of exchanged information, we use a digital signature to verify if received messages comply with sent ones and to identify the source of such messages.

5 Delivery Manager

The information distributed through the P/S middleware adopts the information model presented in Section 4. This is to allow for the integration of heterogeneous registries and also to support the creation of special-purpose filters to retrieve services efficiently. Heterogeneity and filters are managed by the *delivery manager*, that is, a *facade* element that standardizes the information flow. The delivery manager is responsible for the policies for both sharing the services in the registry and selecting the services that must be retrieved from the P/S infrastructure. Its adoption does not require modifications to the publication and discovery processes used by the different organizations behind the registries. They keep interacting with the repository they were used to, but published services are distributed through the P/S infrastructure, which in turn provides information about the services published by the others.

If adopted technology distinguishes between *registry* and *repository*, where the first contains references to the objects stored in the second, the delivery manager handle this distinction. It is also in charge of managing the federations the registry belongs to and of re-distributing significant data.

Service Publication. Services are published by providing the *delivery manager* with what should be shared. A company can choose to share data created on its registry or information retrieved from others. In both cases, the owner of

published data remains the only subject able to update them. The manager simply retrieves what has to be published from the registry, transforms it into the right format, and then forwards it onto the P/S infrastructure.

Since *DIRE* is a decoupled distributed system, a registry can join and declare its interests at any time. The infrastructure guarantees that a registry can always retrieve the information it is interested in. For this purpose, the propagation of information is subject to *lease* contracts, a typical concept of many distributed systems (e.g., Jini [10]). When the lease expires, the information is not considered to be valid anymore; only a *renew*, which requires that the information be retransmitted, allows for extending its validity. The delivery manager can perform this operation automatically to guarantee that when the information is re-sent, all interested registries retrieve it.

The lease period τ is configurable at run-time; it guarantees that the information about services are re-trasmetted with a user-defined frequency. This means that τ is the maximum delay with which a registry is notified about a service. Moreover, if the description of a service changes, the lease guarantees that the new data are distributed to all subscribed registries within τ time units.

Service Selection. Because of commercial agreements between the parties, a client may know in advance the services it wants. In this case, the selection can be precise: the unique identifiers of interested elements are used by the delivery manager to create a filter and subscribe to them through the dispatcher. Since published information is refreshed periodically, the result is that all relevant data about selected services are stored in the client's registry.

When the client does not know the services it wants, *DIRE* allows the client to retrieve all the services whose descriptions comply with specified properties. If the property is encoded in a StandardFacet, the client can use the unique identifier of the facet and select all the services that have a Relationship with the selected property (facet). If the property is defined through a SpecificFacet, the client can express its requirements by using an XPath expression; the delivery manager wraps it into a filter and passes it to the dispatcher to subscribe to "relevant" services. This allows the delivery manager to receive the service descriptions that match desired properties. The delivery manager receives the unique identifiers associated with retrieved services and automatically create the filters to get them.

Notice that since the information is periodically re-transmitted, *DIRE* does not require any direct interaction between the registry that sends the information and those interested in it. Moreover, once data are received from other registries, the receiving actor (registry) cannot update them; it can only attach new SpecificFacets to further characterize retrieved services.

Federation Management. *DIRE* federates registries to allow them to "share" services. Each member of a federation can re-distribute the information it gets about a service to every registry of the federation. A registry can promote both proprietary services and services received from others.

Federations are treated as special-purpose subscriptions. Usually, we adopt *content-based* subscriptions, but federations exploit *topic-based* subscriptions.

Each federation is associated with a *topic* (that becomes the name of the federation). When a registry joins a federation, it must subscribe to the associated topic. This ensures that every time there is a message for that topic, it is received by all the participants of the federation, and thus we have a multicast communication group. Notice that if there are updates on services promoted in a federation, the registry that initially shared them must re-send the new information to the entire federation.

6 Case Study

This section introduces a simple case study to clarify how delivery managers help registries exchange information. The scenario, shown in Figure 2, considers four actors: *MyAirline*, *BigCompany*, *SmallCompany* and *TinyCompany*. The last two are subsidiaries of *BigCompany*: they are independent but federated entities, which need to efficiently communicate with the holding company. On the technical side, *MyAirline* has a UDDI registry, while the others use ebXML registries. In this context, *DIRE* fosters the collaboration between *BigCompany* and its subsidiaries, and enables the communication between *MyAirline* and its potential clients.

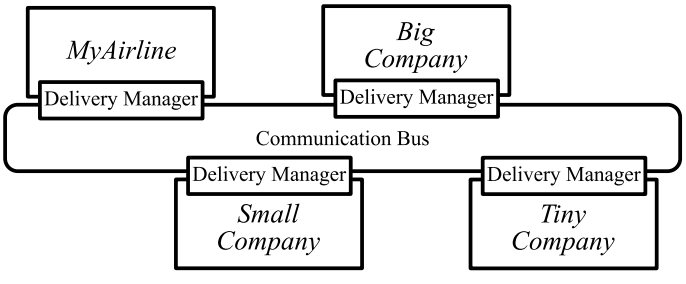

Fig. 2. Architecture of the scenario

MyAirline acts as service provider, and has two main services: one to compute the salaries of its employees and the other to sell airplane tickets on-line. The company wants to keep the first service private, while it wants to disseminate the second: if more companies discover the availability of such a service, *MyAirline* has the opportunity to increase its revenues.

As for the model presented in Section 4, we create one Organization element to describe the company (*MyAirline*), one Service to publish the online reservation service, and one ServiceBinding for each access point available for the offered service. To better advertise the service, *MyAirline* characterizes it through a set of facets. We use two StandardFacets, to identify the WSDL interface of the generic service, and to specify the ISO 9001 quality level of the company. Some SpecificFacets define the business data —like the semantic categorization, required commissions, and payment options— and identify the adopted transport

protocol, encoding, and quality of service for each ServiceBinding. The relationship with the ISO 9001 facet is signed by an external company, which acts as certification authority, and thus the receivers can rely on its authenticity.

All this information (i.e., all these facets) are stored in the registry controlled by *MyAirline*. When the company decides to share the reservation service, the delivery manager can easily convert these data into our technology-agnostic format and publish them on the P/S infrastructure. Since *MyAirline* does not want to share the service to compute the salaries of its employees, it is kept private and no information is published. After the first publication, the delivery manager periodically re-sends the information about the service through the communication bus (i.e., through the *ReDS* dispatcher).

On the other side, *BigCompany* wants to optimize the purchase of airplane tickets. If *BigCompany* knows that *MyAirline* has a service that might be useful, it can subscribe to it directly. Otherwise, *BigCompany* can look for a service that complies with the interface of a generic online flight reservation service. As explained above, this property is rendered through a StandardFacet, and thus its unique identifier can be used to retrieve the set of services associated with that facet. As last option, *BigCompany* can exploit the semantic information associated with services to retrieve what it needs. It can create an XPath expression to search the content of RDF facets. All the options produce a subscription filter that embeds the requirements: when *MyAirline* renews the lease of the information about its service, *DIRE* sends all the elements that comply with the request (filter) to *BigCompany*.

BigCompany is happy with the service provided by *MyAirline* and wants to propagate the use of this service to its subsidiaries. To do this, *BigCompany* exploits the federation *BigCompanySubsidiaries* to propagate the information about the interesting service. The subsidiaries become aware of the new service, retrieve its data, and store them in their registries. The service supplied by *MyAirline* becomes available to the whole set of registries.

6.1 Experimental Assessment

The approaches described in Section 2 require that clients and information provider (i.e. *BigCompany* and *MyAirline* in our scenario) interact directly. Our methodology fully decouples the cooperation among registries, but requires the adoption of the lease mechanism: there is a periodical re-transmission of shared data, thus the exchange of messages, which is higher than with direct cooperation, might become the bottleneck of the whole approach.

Even if the use of *ReDS* in other domains gave encouraging results, we decided to conduct some empirical studies to assess the actual performance of *DIRE*. To analyze the scalability of the approach, we decided to particularly stress the communication bus and the delivery manager. In our tests, we exchanged 170,000 messages, that is, more than three times the entries in the UBR before its shutdown. Table 1 summarizes the results on the performance of the communication bus when we use simple filters, based on unique identifiers, and complex XPath ones. The 150,000 messages of the first column are handled with a mean time of

Table 1. Performances of the communication bus

	Simple filters	**XPath filters**
# of publications	150,000	20,000
# of subscriptions	40,000	10,000
Computation time	1 min 23 sec	9 min 04 sec
Mean time	0.55 ms	27.27 ms
Throughput	6,500,000 msg/h	132,000 msg/h

Table 2. Performances of the delivery manager

	New elements	**Renewed elements**
# of messages	50,000 msg	100,000 msg
Computation time	33 min 18 sec	26 min 3 sec
Mean time	39.96 ms	15.63 ms
Throughput	90,000 msg/h	230,000 msg/h
Standard deviation	6.32 ms	1.99 ms
99^{th} percentile	76 ms	24 ms

0.55 ms, and thus a throughput of 6,500,000 publications per hour. The 20,000 messages of the second column are managed in 9.04 min, which implies a mean time of 27.27 ms. These figures highlight good results for managing simple filters and acceptable ones for complex XPath expressions.

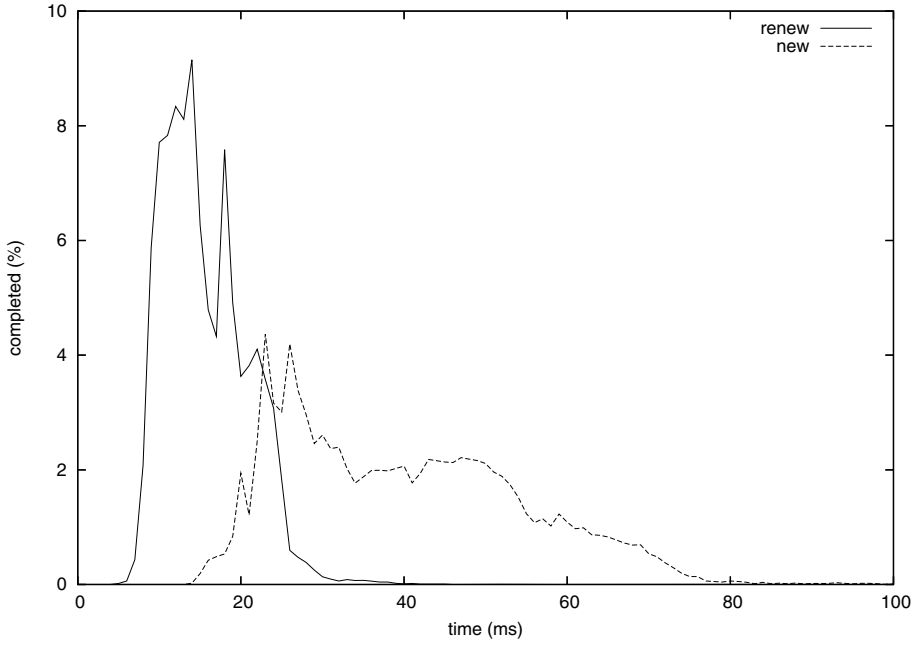

Fig. 3. Percentage of processed messages with respect to elaboration time

As for the delivery manager, we noticed that different messages impose different delays. Figure 3 plots the percentage of processed messages (y axis) with respect to elaboration time (x axis). There is a clear difference between the first time the delivery manager receives an element (dashed line) and subsequences renews (solid line). In particular, our experiments (Table 2) show a mean processing time of 39.96 ms for a new message and of 15.63 ms for a renew. The 99^{th} percentile, that is, an estimation of the maximum time required to process a message, is very low (76 ms for a new element, 24 ms for e renewed one), and this indicates good global performance for the delivery manager. The low computational time required for processing renewed elements also demonstrates that the lease mechanism does not introduce a significant overhead. Our experiments suggested that a two-hour renew period is a good trade-off between a fast lookup and a non-saturated system. This threshold seems to be a very reasonable compromise for the effective deployment of *DIRE* in real settings.

7 Conclusions and Future Work

The paper presents *DIRE*, a framework for the cooperation and federation of distributed and heterogeneous registries based on the publish/subscribe paradigm. The *dispatcher* acts as common reference for the registries that want to communicate. Each entity is free to decide what information —and thus what services— it wants to share within the community by *publishing* it through the dispatcher. Similarly, registries can also decide the services they are interested in by *subscribing* to particular service types. Federations can be set among registries to support the broadcast of information among the elements that belong to the federation. *DIRE* also proposes a dedicated service model to provide powerful and flexible descriptions of services and to support the creation of powerful filters for sophisticated subscriptions. The proposed model is independent of the technology of the registries that form the community.

Our future work comprises the adoption of the *publish-subscribe-reply* paradigm, to increase the reactivity of the framework, the extension of our current interaction model, to support the direct push of significant information about newly published services towards the end users, and more attention to the confidentiality of published information, to avoid that reserved data are used by unauthorized entities. We are also working on better distinguishing between service specification (i.e., facets created by the service provider) and service additional information (i.e., facets created by the users).

References

1. Peter Brittenham, Francisco Cubera, Dave Ehnebuske, and Steve Graham. Understanding WSDL in a UDDI registry. `http://www-128.ibm.com/developerworks/webservices/library/ws-wsdl/`.
2. David Chappel and Tyler Jewell. *Java Web Services.* O'reilly, 2002.
3. Gianpaolo Cugola and Gian Pietro Picco. ReDS: A Reconfigurable Dispatching System. `zeus.elet.polimi.it/reds`.

4. Francisco Curbera, David Ehnebuske, and Dan Rogers. Using WSDL in a UDDI registry.
5. ebXML. ebXML: Electronic Business using eXtensible Markup Language. `http://www.ebxml.org/`.
6. IBM. UDDI Business Registry shutdown FAQ. `http://www-306.ibm.com/software/solutions/webservices/uddi/`.
7. T. Pilioura, G. Kapos, and A. Tsalgatidou. PYRAMID-S: A scalable infrastructure for semantic web services publication and discovery. In *RIDE-DGS 2004 14th International Workshop on Research Issues on Data Engineering, in conjunction with the IEEE Conference on Data Engineering (ICDE 2004)*, March 2004.
8. Cristina Schmidt and Manish Parashar. A peer-to-peer approach to web service discovery. *World Wide Web*, 7(2):211–229, June 2004.
9. Sun. Java Api for Xml Registries. `http://www.jcp.org/en/jsr/detail?id=93`.
10. Sun. Jini. `http://www.jini.org/`.
11. UDDI.org. Universal Description, Discovery and Integration version 2.0.
12. UDDI.org. Universal Description, Discovery and Integration version 3.0.2, October 2004.
13. Kunal Verma, Kaarthik Sivashanmugam, Amit Sheth, Abhijit Patil, Swapna Oundhakar, and John Miller. METEOR-S WSDI: A scalable P2P infrastructure of registries for semantic publication and discovery of web services. In *Information Technology and Management*, volume 6(1), pages 17 – 39, Jan 2005.