OSDA: Open Service Discovery Architecture for Cross-domain Service Discovery

Noura LIMAM*, Joanna ZIEMBICKI*, Reaz AHMED*, Youssef IRAQI*, Dennis Tianshu LI*,
Raouf BOUTABA*, Fernando CUERVO†
*University of Waterloo. Waterloo, Ontario. Canada.
†Alcatel Interworking, Inc. Ottawa. Canada.

***Abstract*— Emerging service-oriented architectures are pushing towards on-demand and "on the fly" application and business process composition. In order to support service composition, the underlying infrastructure must provide a facility for on-demand discovery of services and service components. Discovery becomes challenging when services span different networks and/or discovery domains. For inter-domain discovery to be achieved independently of domain-specific service discovery technologies, a middleware is needed to interface between the different discovery systems. In this paper, we present a novel Open Service Discovery Architecture (OSDA) designed to serve as an efficient, scalable, and programmable middleware for cross-domain discovery. We demonstrate the implementation of OSDA using a set of mature technologies.**

I. Introduction

The emergence of service-oriented architectures for enabling the composition of applications, business processes and services, creates a rich breeding ground for widely-distributed applications that span heterogeneous networks and administrative domains.

Composing services on the fly requires an underlying service infrastructure that provides devices and service components with the ability to discover each other. Service discovery technologies (e.g. SLP, Jini, UPnP, etc.) are good potential candidates to serve as a basis for such an infrastructure. However, the continuing proliferation of these inherently non-interoperable technologies makes interaction between devices and service components increasingly difficult, and hence service discovery more complex.

The heterogeneity of competing discovery technologies creates an incentive for providing a middleware that would enable interactions between service discovery systems. Such a middleware would allow network components under one domain to discover components in other domains, while each domain is controlled by a different service discovery technology.

Our proposal, the Open Service Discovery Architecture (OSDA) federates service discovery technologies and provides a model for cross-technology service discovery. OSDA achieves this goal by establishing an inter-domain model for distributed information storage and querying, as well as a unified information representation.

The remainder of the paper is organized as follows. Section II summarizes related works and discusses the motivation behind OSDA. In Section III we describe the high-level architecture design, while section IV presents a detailed specification of the OSDA components. In the following Section V we present our choice of implementation technologies and report on the status of our implementation. Section VI concludes the paper, discussing unsolved issues and future works.

II. Related works

Over the past few years, many service discovery approaches have been proposed by academia (INS [1], INS/Twine [2], SSDS [3], Splendor [4] etc.) and industrial standardization bodies (Bluetooth SDP [5], SLP [6], UPnP [7], Jini [8], Salutation [9], UDDI [10]). Although they provide the same basic functionality of service discovery, they differ significantly in architecture, message exchange pattern, expected operating environment (e.g. mobile vs stationary services, real-world vs. web services, etc.), and service representation/description. These differences make their interoperation difficult.

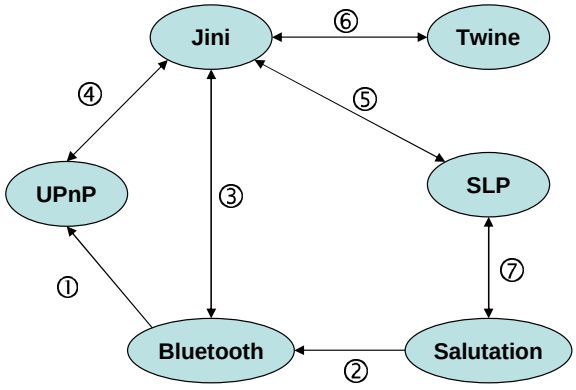

Fig. 1. Existing approaches for interworking service discovery technologies: (1) Bluetooth-UPnP [11], (2) Bluetooth-Salutation [12], (3) Bluetooth-Jini [13], (4) Jini-UPnP [14], (5) Jini-SLP [15], (6) Jini-Twine [16], (7) Salutation-SLP [17]

As shown in Figure 1, a number of works ([11], [12], [13], [14], [15], [16] and [17]) have been conducted to enable the interaction between two different service discovery technologies. Each of these work employs a kind of "bridge" for protocol and data conversion.

Koponen et. al. [15] have presented an architecture for Jini and SLP interoperability. At the core of this architecture are a service broker and an adapter. The adapter has two-fold functionality: it acts as directory service (i.e. Directory agent for SLP and Lookup service for Jini) and it registers services in other domains with the local directory service. An adapter captures local advertisements and forwards them to

the broker. The broker in turn registers these advertisements with the directory service of each domain using the adapter in respective domain. A client can discover services in remote domains, simply by querying its local directory service. This approach is not suitable for networks with a large number of domains, due to two reasons. First, all advertisements are mirrored in the directory service of each domain, which raises a scalability issue. Second, the broker is a single point of failure and performance bottleneck.

Another work [14] presents an architecture for interworking Jini and UPnP, where virtual clients and services are placed in each domain. For a service that is discovered by a virtual client in one domain, a corresponding virtual service is created in the other domain. The virtual service registers itself to Jini Lookup Service (in Jini domain) or multicasts its existence (in UPnP domain). A client can discover and access a service in a remote domain using the virtual service present in its own domain. This approach is not efficient for connecting a large number of domains as all the services of all domains are mirrored in each domain.

A different approach [16] for interworking Jini and Twine adds a proxy component between both domains. The proxy acts as a lookup service in the Jini domain and both as a client and service in the Twine domain. It forwards both advertisements and queries coming form the Jini domain to the Twine domain. Hence, Jini services are registered in the Twine domain, and while queries coming from Jini clients are solved both in the Jini and Twine domains, queries coming from Twine clients are resolved only in the Twine domain. Applying such an approach to different discovery systems, for instance UPnP instead of Twine, would assume that both domains are part of the same network, for instance local area network. Such an assumption is not suitable for service discovery in wide-area networks.

In contrast to all the existing approaches to bridging service discovery systems, OSDA can operate on multiple domains with diverse discovery mechanisms, and can support a large number of services/users participating over a wide-area-network. For cross-technology service discovery, OSDA uses an open and interoperable service description scheme. OSDA is designed to be platform-independent, extensible and fault tolerant, and provides straight-forward ways to introduce access control policies.

III. Open Service Discovery Architecture

In the remainder of the paper, a "domain" is defined as a federation of network components (users and services) controlled by a single service discovery technology. The main motivation of our work is the need to federate users and services spanning different domains whether they belong to the same or different networks. To this end, we build on the existing domain-specific discovery systems by providing the following facilities:

- as an alternative to mirroring shared services or broadcasting queries in all the involved domains and networks (which do not scale with an increasing number of domains and services), a **peer-to-peer indexing overlay** is created as an inter-domain and inter-network space where shared services are advertised and queries are solved.
- programmable **brokers** are deployed in domains to act as an interface between the intra-domain and inter-domain discovery systems.
- as an alternative to converting service advertisements to all involved service description schemes, a **unified service description scheme** is used for the advertisement of services in the inter-domain space.

We design our system as a pluggable solution that does not require any change in the local discovery systems and that is extensible to new participating domains.

A. High-level Architecture

In Figure 2 we present a high-level overview of OSDA. We assume that service discovery components (user agents, service agents and eventually service registries) are already in place. Moreover, domains may deploy *Policy Servers* for controlling the propagation of service advertisements and service queries outside the domain boundaries.

As stated before, we introduce in each domain one or more than one service brokers and build a peer-to-peer indexing network in the inter-domain space. The canonical setup would involve one broker per domain and a one-to-one association between brokers and peer indexing nodes, however, the architecture supports multiple brokers per domain and one-to-many, also many-to-one, association between brokers and peers for the sake of tolerance to broker and peer failures.

The following components are integral parts of our system:

- **Service Broker**: responsible for handling and processing cross-domain advertisements and queries. It acts as an interface between the local discovery systems and the inter-domain discovery system. The broker can be divided into two layers; a technology-dependent lower layer and a technology-independent upper layer.
  The technology-dependent layer (referred to as *Adapter*) is the programmable part of OSDA. It abstracts the local service discovery system by intercepting and processing requests coming from User Agents and Service Agents and converting advertisements and queries to a well-defined service description and query language.
  The technology-independent layer handles broker-to-peer and broker-to-broker communication. It provides the necessary interfaces for the broker to be accessed by the entities involved in the inter-domain discovery process.
  Note: Since there is a fairly clear separation between the technology-dependent and the technology-independent layers of the broker, we will occasionally and respectively refer to them as two separate entities: the *Adapter* and the *Broker*.
- **Peer-to-peer Indexing Node**: is responsible for distributing service information in the peer-to-peer overlay and allowing inter-domain discovery. The peer-to-peer network uses a Distributed Hash Table (DHT)-based architecture to store service information and solve queries.

Our system supports the two main functions of a service discovery system: *advertisement* and *querying*. A brief outline

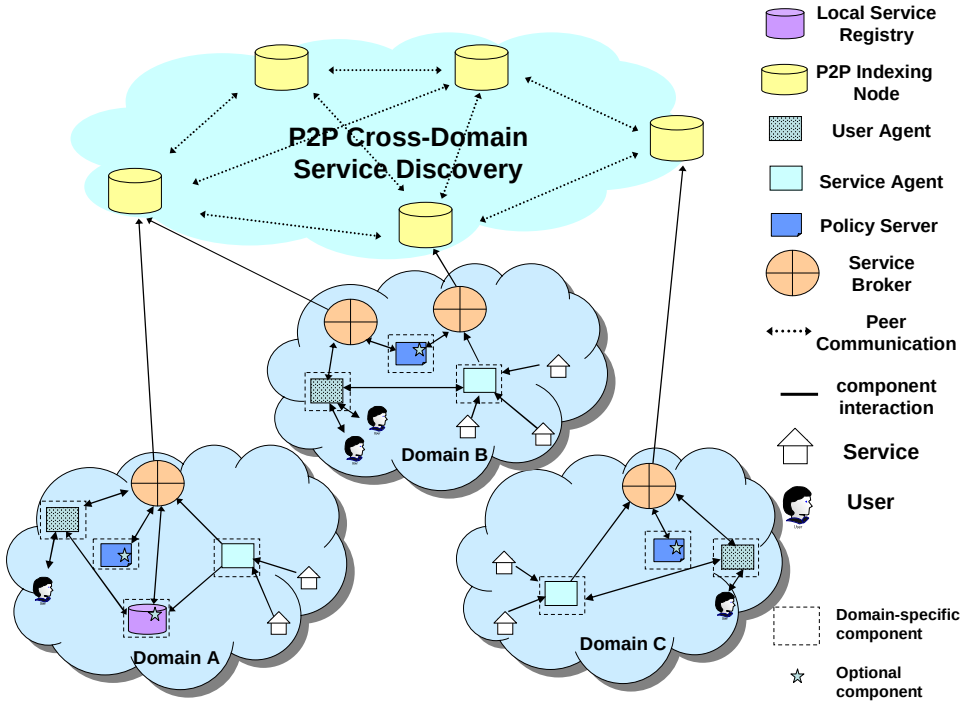

Fig. 2. OSDA: High-Level Architecture

of these functions is given below, while a more detailed description can be found in Section IV.

- Advertisement: The local advertisements that are allowed by domain policies to be propagated in the inter-domain system, are forwarded to the broker after being translated into a common, well-defined service description format. The broker then sends advertisements to the peer-to-peer overlay, which distributes the service information among selected nodes. Only a subset of the service advertisement is propagated to the peer nodes; typically a set of descriptive attributes of the service capabilities along with the URL of the domain's broker(s) responsible for processing queries that relate to the advertised service.
- Querying: Querying in OSDA is a two-step process. First, the query is translated to a common format, and forwarded by the broker to the peer-to-peer overlay. The broker receives back the set of broker URLs that have been advertised along with the services matching the query. Second, this same broker contacts one or more brokers from the received list in order to retrieve the entire information about the requested service, especially its access point. This second step may require the contacted broker to execute queries in the local discovery system. Splitting up the querying process in this way allows the domains to control which parts of service information are advertised to the world at large, and which parts are made available only to selected domains. It also allows the domains to control the amount of data forwarded to the peer-to-peer overlay by sending a single inter-domain advertisement for a set of similar services.

The resulting architecture acts as a unifying "glue" that connects diverse service discovery systems.

B. Service Description Scheme

Each service discovery system has its own way of describing a service. For supporting interoperability among a variety of service discovery systems, some means of vocabulary translation is needed. Vocabulary translation can be carried out directly from one technology to the other: a Jini advertisement (or service description) can be converted to an equivalent SLP advertisement and advertised in the SLP system. However such a scheme would require $O(N^2)$ mappings, where N is the number of supported service discovery technologies. Clearly, the preferred method would be to design an intermediate, unified scheme for inter-domain advertisements. In this case, an advertisement from a particular domain can be translated to the agreed inter-domain format and can then be advertised in an other technology after another step of conversion. Because of its expressive power and acceptance in the Internet community, XML seems to be most appropriate as a basis for such a format, while the selection of appropriate elements for the XML description needs more detailed analysis of the existing service discovery technologies.

To achieve this kind of interoperability, we propose a service description schema that can interoperate with most of the major service discovery systems. In this section we present the Unified Service Description (USD), a schema from which service templates can be derived and used as the commonly agreed service templates.

It is worth noting that unlike the Web Service Description Language (WSDL) used to describe and publish Web Services in the UDDI system, the USD scheme can be mapped to any service description scheme. Given that WSDL is restricted to the representation of service interfaces and binding information, it is definitely not suitable for carrying information about

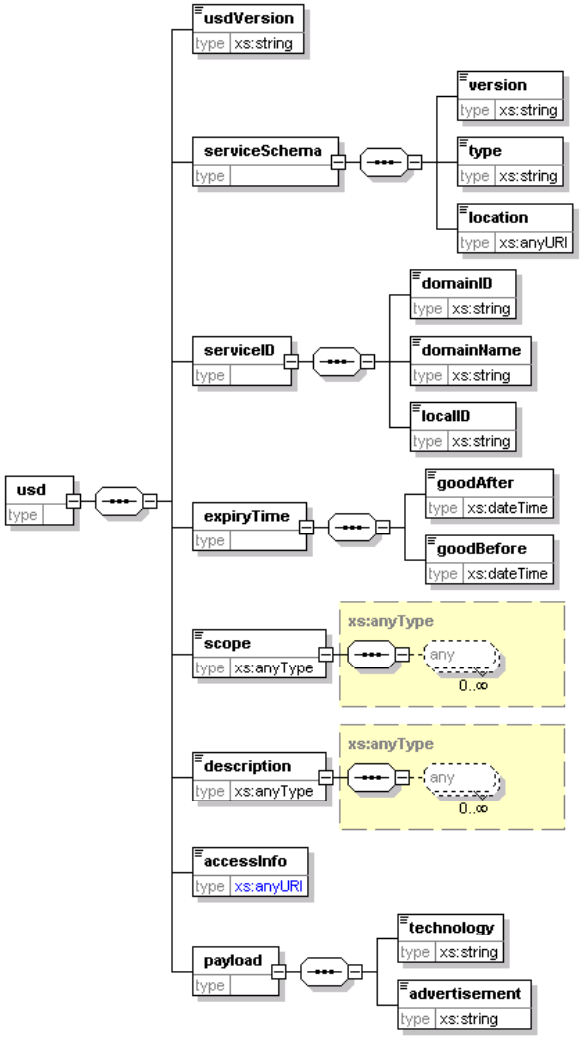

Fig. 3. USD: Unified Service Description

service capabilities as required by other systems like SLP, Jini, Twine, etc.

As shown in Figure 3, the USD scheme consists of two major nested parts. The main envelope contains the meta-information for the advertisement, such as the type, location, expiry time and access information of the service. The *description* component specifies the properties and capabilities of the service itself.

Table I summarizes the fields contained in the USD scheme.

For better understanding of the discovery process, let us focus on two key fields of the Unified Service Description: *serviceID* and *description.*

- serviceID: In OSDA, a service identifier consists typically of two parts: a globally-unique domain identifier, and a domain-unique service identifier. Together, these two parts form a globally-unique identifier for each service, obviating the need for a central naming authority that assigns names to each participating service. The format of the service identifier is left up to the domain administrator, giving each domain discretion in naming its services. The domain identifier is more difficult to define, since it must be globally unique. In the case where "domain" means simply "the administrative domain", we recommend the use of DNS names. In other cases, uniqueness can be accomplished by using a Universal Unique Identifier (UUID) [18]. If a non-human-readable domain identifier is used, such as the UUID, we recommend the use of the *domainName* field to include a human-readable domain name with the USD.
- description: OSDA supports any description schema (or service template) provided that it is written in XML. Typically, service templates are generated by service providers and may be commonly used by a number of service providers as the "standard" template for inter-domain advertisements.

We additionally provide XML-based message formats for global service discovery requests and responses.

IV. Module Specification

As described in Section III, OSDA is comprised of several components. Each of these components consists, in turn, of a number of smaller modules. The high-level objective of

Field	Description
usdVersion	Specifies the USD version from which the service template was derived
serviceSchema	Refers to the XML Schema Definition (XSD) which serves as a template for the service description. **Version**: specifies the version of the service template. It allows incremental upgrading of service descriptions with backward compatibility **Type**: a gross category of the service referring to the service template **Location**: a URI specifying the location of the service template XSD document
serviceID	The service identifier used to globally and uniquely identify a service. It contains the following information: **domainID**: unique identifier for the domain (can be non-human readable) **domainName**: human-friendly domain name supplied by the system administrator **localID**: the name used to uniquely identify a service within a domain
expiryTime	The time when an advertisement expires. It consists of two fields: **goodAfter**: time by which the advertisement starts to be valid **goodBefore**: time by which the advertisement is no longer valid
scope	may contain (a reference) the list of domains that are allowed to discover the service. It allows domain-level access control for broker-to-broker communication
description	The capability description of a service. It is a set of attribute-value pairs in a hierarchical relationship. We propose the use of XML Schema Definition (XSD) for describing the capability template (analogous to the service template in SLP or UPnP).
accessInfo	A URL that can be used to invoke a service. The URL may point to a static document (e.g. WSDL document) that describes the interfaces to access the service, or embed the request needed for obtaining the service access point. The URL may also point to an intermediate entity (e.g. proxy) that mediates the invocation syntax and semantics.
payload	Used to provide the client with the original description of a discovered service as advertised in its domain. The payload information consists of two components: **technology** refers to the local discovery technology (e.g. Jini, SLP, etc.) **advertisement** the description of the service as advertised in its domain The payload information may allow a client aware of a specific technology to perform technology-specific operations on the discovered service.

TABLE I
UNIFIED SERVICE DESCRIPTION

the design of OSDA modules is maximizing the following properties:

- Modularity: clean separation of responsibilities between components, allowing a reduced maintenance effort,
- Scalability: support an increasing number of service advertisement and discovery requests across multiple domains,
- Reusability: clean identification of functionalities for each component in order to reduce code duplication and to create reusable components.

In the following section, we will examine the module-level design of OSDA (see Figure 4), by explaining the functionalities and interfaces of the individual modules.

A. Adapter modules

The Adapter is composed of the following four modules: the *Registration Advertisement Handler*, the *Discovery Request Handler*, the *Directory Handler* and the *Converter*.

1) **Registration Advertisement Handler**
The Registration Advertisement Handler is responsible for processing advertisement requests coming from local Service Agents. Upon interception of an advertisement, the Registration Advertisement Handler contacts the policy server (if present) so that related domain policies are applied. If the advertisement is allowed to be propagated in the inter-domain discovery system then the Advertisement Registration Handler first converts it to the USD format and then submits it to the Broker's Advertisement Propagator.
This process is illustrated in Figure 5. Note that in this example, the policing step is omitted.

2) **Discovery Request Handler**
This module is responsible for intercepting and handling service discovery requests. It steers queries either to the local or global discovery systems according to the the domain policies. If a service discovery request is allowed to be propagated to the inter-domain discovery system, it is first converted to the USD-based query format, and then submitted to the Broker's Global Discovery Handler (see Figure 6). Similarly, upon reception of a response to a query, the response is first converted to the local description format and then forwarded to the User Agent that generated the query (see Figure 7).

3) **Directory Handler**
The Directory Handler is responsible for handling Broker's service discovery requests in the second step of the querying process (see Section III). It takes care of converting queries into the local query

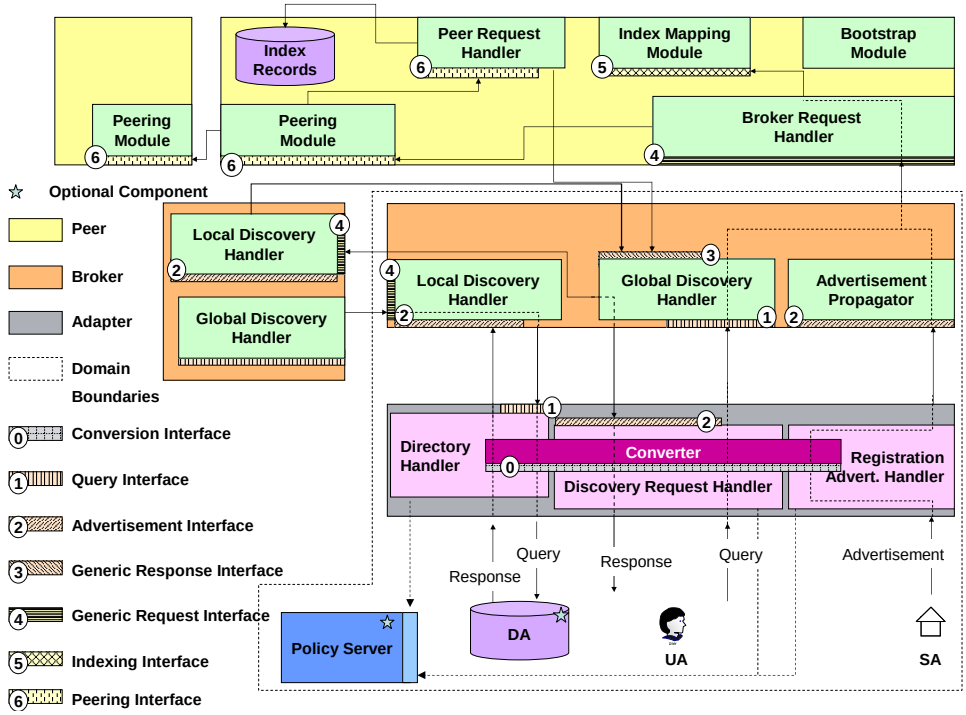

Fig. 4. OSDA: Architecture Specification

format and generating discovery requests in the local discovery system (see Figure 7). The Directory Handler implements a User Agent interface; generated queries are either sent to the service repository or multicasted/brodcasted over the network depending on the discovery mechanism deployed in the domain.

4) **Converter**
The Converter module is used to convert queries and service descriptions into a USD-based format, or from the USD-format to the local format. A mapping between local service templates and USD templates as well as a mapping between the local query format and the Unified Query format are both required in the conversion process.

B. Broker modules

The Broker is responsible for handling cross-domain advertisement and service discovery requests. It provides well-defined interfaces that are invoked to post requests and responses. The broker is composed of the following three modules: *Advertisement Propagator*, *Global Discovery Handler* and *Local Discovery Handler*.

1) **Advertisement Propagator**
The Advertisement Propagator is responsible for propagating advertisements to a peer-to-peer indexing node (see Figure 5).

2) **Global Discovery Handler**
This module is responsible for processing inter-domain service discovery requests. As stated in Section III, inter-domain service discovery is achieved in two steps. In the first step (see Figure 6), the Global Discovery Handler is responsible of the propagation of service discovery requests to the peer-to-peer network. In the next step (see Figure 7), after receiving a list partial service descriptions with associated brokers, the Global Discovery Handler sends the same request to one or more than one broker is the list. A selection mechanism over the received list may be implemented. Responses from the contacted brokers will contain the USD of services that match the discovery request.
It is worth noting that the number of contacted brokers influences both the inter-domain query overhead and the completeness of the query result.

3) **Local Discovery Handler**
The Local Discovery Handler is responsible for locally processing the requests sent by remote Global Discovery Handlers. It sends queries embedded in the received requests to the Directory Handler in order to resolve them into the USDs of matching services. USDs are sent back to the Global Discovery Handler that generated the request.

C. Peer node modules

The network of OSDA peer nodes forms a distributed hash table. The nodes are responsible for storing index records corresponding to previous inter-domain advertisements, and for solving cross-domain queries. A peer node is composed of a *Broker Request Handler*, an *Index Mapping Module*, a *Peering Module*, a *Peer Request Handler* and a *Bootstrap*

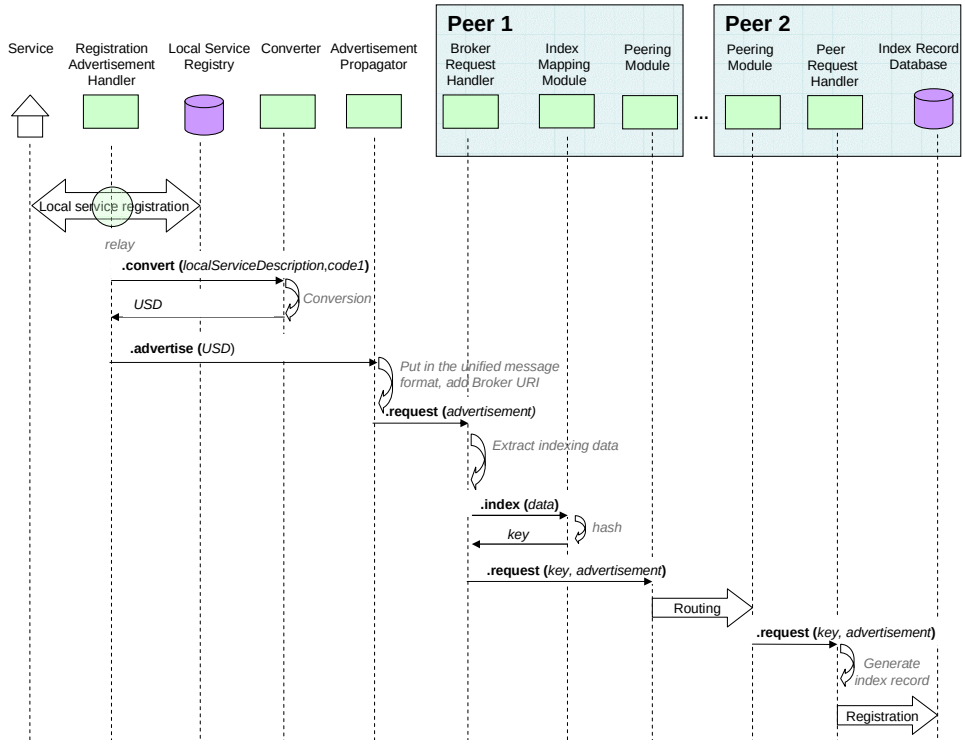

Fig. 5. Sequence Diagram: Service Advertisement

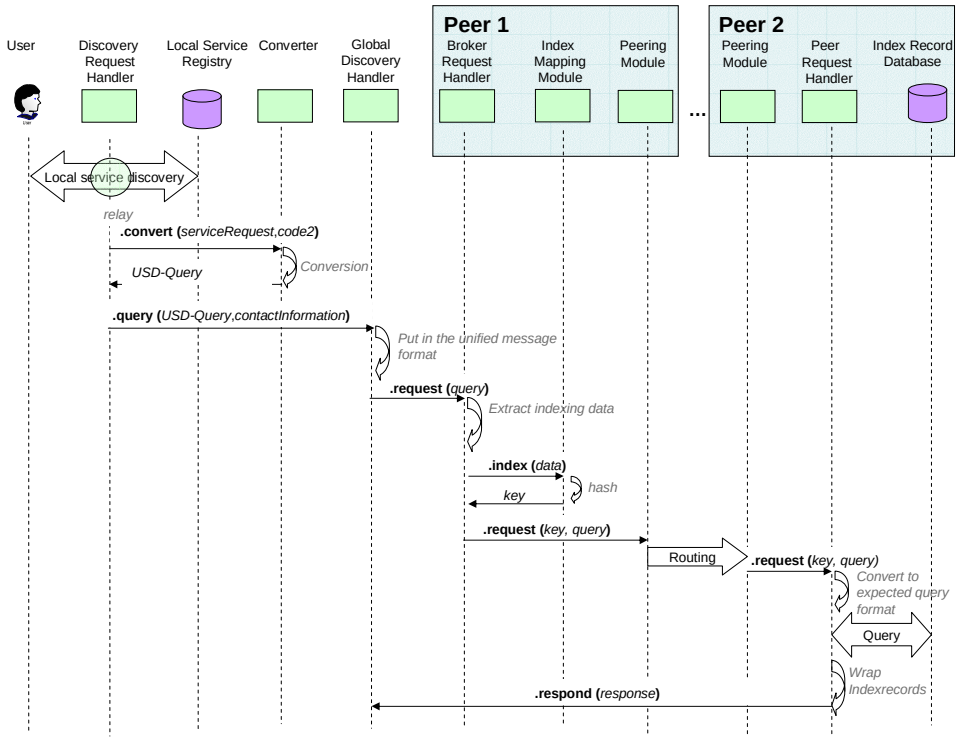

Fig. 6. Sequence Diagram: Service Discovery, Step1

Module.

1) **Broker Request Handler**
The Broker Request Handler is the access point to the peer-to-peer network. It intercepts and processes inter-domain advertisement and query requests sent by Brokers and then directs them in the peer-to-peer network.

2) **Index Mapping Module**

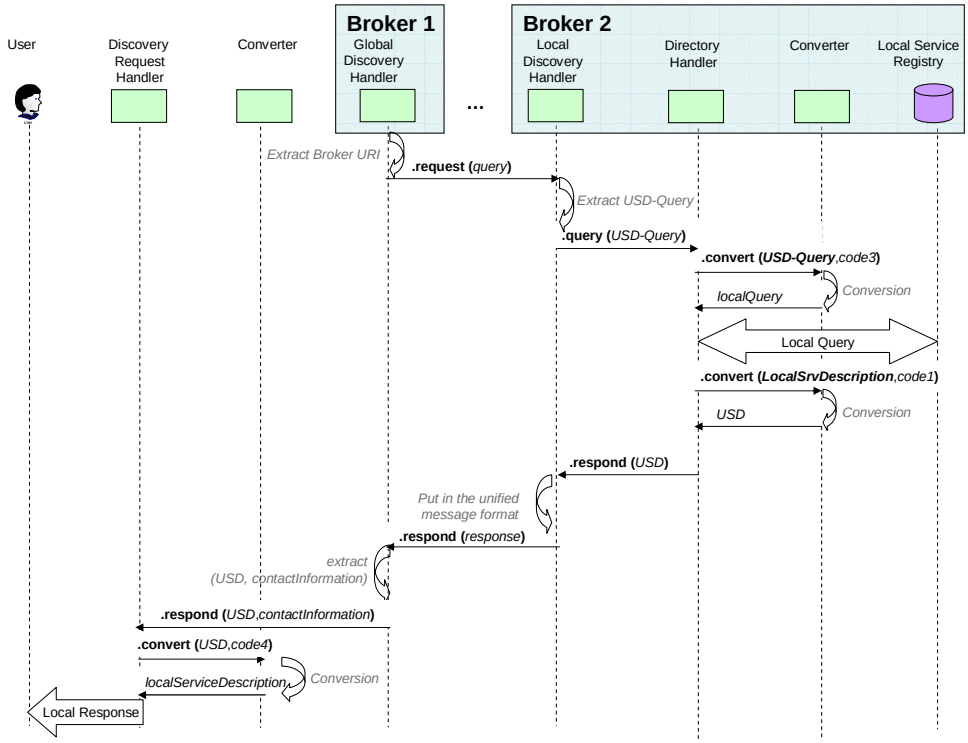

Fig. 7. Sequence Diagram: Service Discovery, Step2

The Index Mapping Module associates index keys to the data that is stored in the peer-to-peer network using a hash function.

3) **Peering Module**
Each peer in the peer-to-peer overlay is responsible for a set of keys. Upon receiving a key-request pair, the Peering Module routes the request to the appropriate peer. If the given key is one of this peer's own keys, the key-request pair is passed to the local Peer Request Handler.

4) **Peer Request Handler**
A peer-to-peer node maintains a database which associates each hash key with one or more than one index records.
Upon receiving a request-key pair, the Peer Request Handler either executes the requested query or stores the requested advertisement in the index record database within the set of index records associated with the given key.

The advertisements stored in the peer-to-peer network are soft-state: each stored index record contains the information about its lifetime. While the advertisement renewal process is handled by the Discovery Request Handler in the domain level, cleaning the peer database from expired index records must be handled by the Peer Request Handler if not supported by the used database management system.

5) **Bootstrap Module**
The Bootstrap Module is responsible for joining the peer-to-peer network. The join process is specific to the DHT architecture used in the peer-to-peer network.

It is worth noting that the inter-component communication is asynchronous and the modules are stateless, i.e. they do not keep track of current requests. This enhances the system's tolerance to "short" failures or disconnections. For example, if multiple brokers are known by an adapter, they can be used interchangeably if one of them fails. Moreover, since information about the source of the query is embedded in each request, a broker can be restarted or replaced between sending the query and receiving the responses. If a broker goes down while waiting for query responses to arrive, those responses can be sent to an alternate broker (if known), or to the originating broker, if it becomes accessible in the meanwhile.

All OSDA components, and most of the modules are designed to be loosely-bound. Contingent on access-control policies, an adapter can communicate with any broker in its domain (allowing several adapters per broker, or vice-versa), while a broker can communicate with any peer indexing node. Such flexibility and degree of fault tolerance is mandatory, since OSDA is designed to be a core supporting function for an Internet-scale network of resources and services.

V. Implementation and Testbed

A. Implementation Technologies

As the name of our architecture would imply, its main goal is to be as open and universal as possible. We used this motivation as a guide in our choice of tools and technologies

that would implement our system. We focused on well-known, tested, freely available and open-source components such as JXTA [19], Chord [20], JBoss [21], INS/Twine [22], Jetty [23] and web-based technologies such as SOAP. Because of the need for platform independence, we used Java as the programming language, HTTP as the transport protocol for broker-to-broker and broker-to-peer communications and XML as the format for all communications. In addition to greatly simplifying the implementation process (in comparison to re-inventing the communication/data format wheel), using primarily web-based technologies allowed us to create a flexible and modular system. If needed, many of the components, such as the index-record database, or the peer-to-peer routing mechanism can be relatively easily replaced with other technologies. Moreover, because of the loose coupling between components, the individual parts of the system can be flexibly deployed on separate systems or even all on the same machine, allowing OSDA to scale gracefully in face of increasing load.

- **Broker:** The broker functions as a standalone Enterprise Java server. The broker modules are implemented as stateless-session Enterprise Java Beans (EJBs) running in the JBoss application server, and deployed as Web Services. This way, the broker components may be invoked either using RMI/IIOP or SOAP/HTTP. Currently, Local and Global Discovery Handlers are accessed by peers and other brokers, cross domain boundaries, through SOAP/HTTP which offers many attractive advantages like the support of security mechanisms and the ability to work through firewalls.
  One of the most interesting advantages of this implementation is that it does not require a local service discovery system to be deployed in a domain. Because the Global Discovery Handler is implemented as a Web Service, a web client can be easily created and then used to discover services in other domains.
- **Peer Nodes:**
  1) **Broker Request Handler:** The Broker Request Handler is implemented as a SOAP service and runs on top of the lightweight Jetty HTTP server. We use INS/Twine libraries to extract from advertisements and queries the data that will be used to generate indexing keys. The service capabilities part of advertisements is split into strands, using the INS/Twine model, and all strands are hashed into keys by the Index Mapping Module (see Figure 8). Advertisements are routed to and then stored/replicated in each peer corresponding to one of the resulting hash key. On the other hand, queries are forwarded to the peer that is associated to the key resulting of the longest strand.
  2) **Index Mapping Module:** The Index Mapping Module uses an MD5 [24] hash to turn strands into hash keys later used for routing.
  3) **Peering Module:** The Peering Module is implemented as a JXTA peer to take advantage of the bootstrapping, authentication, and secure communication facilities of the JXTA environment. Peering

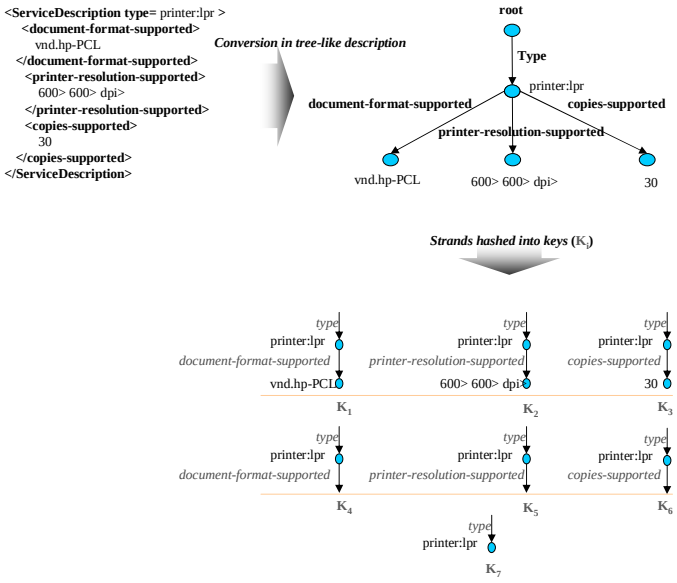

Fig. 8. Advertisement stranding

Modules are organized in a Chord ring, where Chord is used to route requests between peers.
We chose Chord as a base for peer-to-peer network because of its fault-tolerance and self-stabilizing properties as well as its effective method of evenly distributing information and query process load. This design is necessary in dealing with the large volume of advertisement messages and query operations expected in an Internet-scale network. Below, we provide a short analysis of the query costs in the OSDA implementation.
Unlike broadcast-based approaches, the Chord DHT guarantees an upperbound of $O(\log N)$ hops between peers for each routed request, where N is the number of peers in the overlay. After being advertised locally, each advertisement generates k strands, and hence k keys to be distributed among the peer-to-peer nodes, resulting in $O(k \log N)$ messages. The advertisement overhead therefore combines the cost local advertisement with the peer-to-peer advertisement overhead. Additionally, since the peer-to-peer layer uses a soft-state approach, each advertisement must be periodically re-advertised to remain valid, which makes the total overhead dependent on the frequency of advertisement renewal. Because a query generates a single strand, the peer-to-peer query overhead is lower at $O(\log N)$ messages.

4) **Peer Request Handler:** The function of the Peer Request Handler is to store and retrieve XML-based index records to and from a database. For this purpose, we used the eXist [25] open-source native-XML database. Like JXTA, and our SOAP interfaces, eXist relies on the lightweight Jetty HTTP server [23], and includes many useful features such as a web interface and XPath/XQuery [26] process-

ing. While the canonical INS/Twine search would return all service descriptions corresponding to the given key, the Peer Request Handler goes a step further by converting the original query (possibly containing complex predicates – currently, XSet [27] range predicates are supported) into an XQuery. The XQuery is then performed against the index records corresponding to the given key, returning only the relevant set of matching documents.

B. Testbed and Implementation

Currently, our testbed consists of two Linux PCs, each running a peer node, an SLP daemon and a broker. We have implemented an SLP adapter which acts as a virtual SLP Directory Agent for intercepting advertisements and queries coming respectively from users and services. The intercepted messages are then forwarded to both the real Directory Agent and the broker. The SLP adapter also runs a process which intercepts queries coming from the broker at the second step of inter-domain queries. Responses to queries coming both from the local Directory Agent and from the broker are grouped and then forwarded to users. We have also implemented a web portal for issuing advertisements and queries directly to the broker. We use the portal for simulating a third domain.

We have validated the advertisement and inter-domain query processes. Our prototype implementation so far has successfully combined a diverse set of technologies to implement end-to-end inter-domain service discovery.

Because cross-domain service discovery is such a complex topic, we were clearly not able to address all the related issues in the current implementation. For example, in the current implementation of OSDA a broker is statically configured to contact a specific peer in the peer-to-peer network. We intend to work on implementing a mechanism that allows brokers to dynamically find a peer node in the peer-to-peer network. Moreover, we intend to include more sophisticated hash function in the peer-to-peer network that would enable complex and simple advertisements/queries to be handled equally. Currently, complex advertisements and queries are simplified, for existing hash functions do not provide any mapping between the hash of a range-value and the hash of values contained in the range. For example, our DHT does not support matching a query containing $attribute > x$ to a previous advertisement containing $attribute = x + 1$. This functionality is instead provided at the database level, by converting the previously-ignored range query predicates into an XQuery, and using it to narrow down search results.

VI. Conclusions

In this paper, we have presented a new cross-domain service discovery architecture: OSDA. It allows the service providers and consumers seamless access to service and resource discovery across domains using local discovery mechanisms. A previous work [28] that consisted in an in-depth study of the existing service discovery technologies and the requirements for a large-scale, inter-domain service discovery system, has served as guideline to the design of OSDA.

The proposed architecture is designed to be scalable, extensible, efficient and robust. Its interoperation with local discovery mechanisms is enabled through programmable components (message interceptors and translators) and standalone brokers. Because of the loosely coupled nature of OSDA components, the system is able to evolve over time and gracefully recover from faults. The use of DHT-based peer-to-peer overlay for cross-domain discovery guarantees a bounded query response time and can manage large volumes of service advertisement and discovery operations. In our initial system prototyping, we have succeeded in incorporating a set of mature technologies in an end-to-end OSDA implementation. Although the full implementation is not yet complete, the results are promising. Using well-defined standard interfaces (i.e. EJBs, Web Service and JXTA), communication protocols (i.e. SOAP and JXTA end-point communication pipes), and unified data representation (i.e. XML for messages and service descriptions), we can successfully bridge the differences among heterogeneous local discovery systems to provide cross-domain service discovery.

In addition to working on completing the OSDA system implementation, we are currently exploring the use of ontologies in defining and integrating service description vocabularies. Using semantic web tools such as Resource Description Framework (RDF) [29] and Web Ontology Language for Services (OWL-S) [30] can help improve search correctness (by ensuring that the search terms are relevant to the domain of discourse through the use of URIs instead of keywords) and search completeness (by exploiting the use of synonyms and ontological relationships between search terms).

References

[1] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley, "The Design and Implementation of an Intentional Naming System," in *Symposium on Operating Systems Principles*, 1999, pp. 186–201. [Online]. Available: citeseer.nj.nec.com/adjie-winoto99design.html

[2] M. Balazinska, H. Balakrishnan, and D. Karger, "INS/Twine: A scalable peer-to-peer architecture for intentional resource discovery," in *Proceedings of the First International Conference on Pervasive Computing*. Springer-Verlag, 2002, pp. 195–210.

[3] S. E. Czerwinski, B. Y. Zhao, T. D. Hodes, A. D. Joseph, and R. H. Katz, "An architecture for a secure service discovery service," in *Mobile Computing and Networking*, 1999, pp. 24–35. [Online]. Available: citeseer.ist.psu.edu/czerwinski99architecture.html

[4] F. Zhu, M. Mutka, , and L. Ni, "Splendor: A secure, private, and location-aware service discovery protocol supporting mobile services," in *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications (PerCom03)*, March 2003, pp. 235–242. [Online]. Available: http://www.cse.msu.edu/ zhufeng/splendor.pdf

[5] Bluetooth SIG, "Specification of the Bluetooth System, Vol. 1, Core, Rev. 1.1," Bluetooth SIG, Tech. Rep., 2001.

[6] E. Guttman, C. Perkins, J. Veizades, and M. Day, "RFC 2608: Service location protocol, version 2," 1999, status: PROPOSED STANDARD. [Online]. Available: http://www.ietf.org/rfc/

[7] UPnP Forum, "UPnP device architecture 1.0," May 2003. [Online]. Available: http://www.upnp.org/resources/documents/CleanUPnPDA101-20031202s.pdf

[8] Sun Microsystems, "Jini Technology Architectural Overview," Sun Microsystem, Inc, Tech. Rep., 1999. [Online]. Available: http://wwws.sun.com/software/jini/whitepapers/architecture.pdf

[9] Salutation Consortium, "Salutation architecture specification (part-1)," June 1999. [Online]. Available: ftp://ftp.salutation.org/salute/

[10] UDDI Consortium, "UDDI Technical White Paper," 2002. [Online]. Available: http://www.uddi.org/pubs/

[11] B. r. c. Palowireless, "Extended service discovery profile for universal plug & play." [Online]. Available: http://www.palowireless.com/infotooth/tutorial/

[12] B. Miller, “Bluetooth whitepaper, mapping salutation architecture apis to bluetooth service discovery layer,” 1999, document number 1.C.118/1.0, version 1.0. [Online]. Available: http://www.salutation.org/whitepaper/BtoothMapping.PDF
[13] S. Kasper and L. Bhrer, “Jini discovers Bluetooth,” 2002, semester Thesis SA-2002.30, Institut fr Technische Informatik und Kommunikationsnetze. [Online]. Available: http://www.tik.ee.ethz.ch/ beutel/projects/sada/
[14] J. Allard, V. Chinta, S. Gundala, and G. R. III, “Jini meets upnp: An architecture for jini/upnp interoperability,” department of Computer Science, University of New Orleans.
[15] T. Koponen and T. Virtanen, “A service discovery: A service broker approach,” in *Proceedings of the 37th Hawaii International Conference on System Sciences*, 2004. [Online]. Available: http://csdl.computer.org/comp/proceedings/hicss/2004/2056/09/205690284b.pdf
[16] Steven R. Livingstone, “Service Discovery In Pervasive Systems,” 2003, the School of Information Technology and Electrical Engineering, University of Queensland, Autralia. [Online]. Available: http://innovexpo.itee.uq.edu.au/2003/exhibits/s370816/
[17] P. S. Pierre and T. Mori, “Salutation and SLP,” the Salutation Consortium. [Online]. Available: http://www.salutation.org/techtalk/slp.htm
[18] P. J. Leach and R. Salz, “UUIDs and GUIDs,” Feb. 1998, status: INTERNET-DRAFT. [Online]. Available: hegel.ittc.ukans.edu/topics/internet/ internet-drafts/draft-l/ draft-leach-uuids-guids-01.txt
[19] Project JXTA, “JXTA J2SE 2.2.1,” 2004. [Online]. Available: http://platform.jxta.org/java/release/2004Q1_Churrasco/release_note.html
[20] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, “Chord: a scalable peer-to-peer lookup protocol for internet applications,” *IEEE/ACM Transactions on Networking (TON)*, vol. 11, no. 1, pp. 17–32, February 2003.
[21] JBoss Inc., “Professional Open Source from JBoss Inc.” 2004. [Online]. Available: http://www.jboss.com/
[22] MIT, “INS/Twine v2,” 2002. [Online]. Available: http://nms.lcs.mit.edu/software/instwine/ins-2-0.tgz
[23] “Jetty java http servlet server.” [Online]. Available: http://jetty.mortbay.org/jetty/
[24] R.Rivest, “RFC 1321: The MD5 Message-Digest Algorithm,” 1992.
[25] “eXist: Open source native xml database.” [Online]. Available: http://exist.sourceforge.net/
[26] W. W. W. C. W3C, “Xquery,” 2004. [Online]. Available: http://www.w3.org/XML/Query
[27] B. Zhao, “The Xset XML search engine and XBench XML query benchmark,” University of California, Berkeley, Tech. Rep. UCB/CSD-00-1112, 2000. [Online]. Available: citeseer.ist.psu.edu/zhao00xset.html
[28] R. Ahmed, R. Boutaba, F. Cuervo, Y. Iraqi, D. Li, N. Limam, J. Xiao, and J. Ziembicki, “Service Discovery Protocols: A Comparative Study,” to appear in Proceedings of IM 2005 (Application Session), Nice (France), May 15-18, 2005.
[29] G. Klyne and J. J. Carrol, “Resource description framework (RDF): Concepts and abstract syntax,” Feb. 2004. [Online]. Available: http://www.w3.org/TR/rdf-concepts/
[30] D. M. et al, “OWL-S: Semantic markup for web services.” [Online]. Available: http://www.daml.org/services/owl-s/1.1/overview/