4TH QUARTER 2007, VOLUME 9, NO. 4

IEEE COMMUNICATIONS SURVEYS
The Electronic Magazine of Original Peer-Reviewed Survey Articles

www.comsoc.org/pubs/surveys

RESOURCE AND SERVICE DISCOVERY IN LARGE-SCALE MULTI-DOMAIN NETWORKS

REAZ AHMED, NOURA LIMAM, JIN XIAO, YOUSSEF IRAQI, AND RAOUF BOUTABA,

ABSTRACT

With the increasing need for networked applications and distributed resource sharing, there is a strong incentive for an open large-scale service infrastructure operating over multidomain and multi-technology networks. Service discovery, as an essential support function of such an infrastructure, is a crucial current research challenge. Although a few survey papers have been published on this subject, our contribution focuses on comparing and analyzing key discovery approaches in the context of large-scale and multidomain networks. The comparison is conducted based on a set of well-defined criteria, leading to a selection of few approaches that can serve as the guide in designing a global service discovery system for large-scale and multi-technology networks.

The increasing need for networked applications and distributed resource sharing carries a strong incentive for an open large-scale service infrastructure that operates over large-scale networks spanning multiple domains. This trend is emphasized in ongoing research (e.g. Web Services initiative [1]) that aims towards service globalization.

In addition to supporting a large number of services/users participating over a large network, service globalization may require interoperation of diverse discovery systems based in independently-administrated domains. Part of that interoperation is sharing information about deployed services across domain boundaries. There is therefore a need for open and extensible service-discovery framework that can incorporate multiple domains and bridge together heterogeneous service discovery approaches. In this work, we keep in mind the particular need for flexibility and extensibility of discovery mechanisms that would enable such an interoperation.

Service and resource discovery is a crucial research topic, since it is required to support any service infrastructure. A large-scale, multi-domain service infrastructure requires a service discovery system that is open, scalable, robust and efficient to a greater extent than a single-domain system. In this article we survey a number of service discovery approaches and extract those characteristics that make them suitable for this kind of infrastructure. In this manner, we use the work both as general survey and as a guide for the design of a service discovery system.

Our analysis and comparison encompasses several key discovery approaches, including Salutation, SDS, SLP, Jini, INS, UPnP, INS/Twine, JXTA, and Splendor. We also examine discovery mechanisms used in Web Services, Grid Services, and content-sharing peer-to-peer (P2P) systems. To formally structure our analysis and comparison, we first define a set of design criteria as a "measuring stick" for evaluating all of the discovery approaches according to our goal of a large-scale service infrastructure. While exposing the strengths and weaknesses of these approaches, we focus on a selected few approaches that can serve as the starting blueprint for a new discovery mechanism.

A few surveys [2–5] have previously been conducted on service and resource discovery. Some of them have considered discovery approaches such as Bluetooth Service Discovery Protocol [6] and DEAPspace [7]. The former targets pervasive computing environments and, more specifically, interactions between Bluetooth enabled devices in Personal Area Networks. The latter is designed for ad-hoc networks and more specifically single-hop ad-hoc networks, and has been enhanced into Konark [8] to address multi-hop ad-hoc networks. These approaches are not considered in this article, since they are aimed to specific environments and are closely similar to other more significant approaches.

The rest of the article is organized as follows: we define the terminology used in this article. We describe the criteria used for the evaluation and comparison of the different discovery approaches against the requirements of our target infrastructure. We present an overview of these approaches, then we evaluate and compare them with respect to each criterion. This leads to the selection of a number of approaches which, in our opinion, best meet the requirements of a large-scale multidomain environment. We then conclude this document. In Appendix 1, we present a number of distributed peer-to-peer indexing schemes used by a few discovery systems.

Terminology

We define a service as a set of functionalities associated with a process or system that performs a task. We say that the process or system implements the service.

A resource is defined as an entity that is used or acted upon by a process or system. A service functionality takes resources as input.

To further clarify the difference between the two above definitions, a mathematical representation can be used to show the relationship between a service and a resource. We can denote a service by a function f, and a resource by an input variable x. The outcome of the service is then returned as $f(x)$. Notice that a service can be invoked by another service. In this case, the outcome of the service f becomes a resource and is used by another service, say g. This is called service composition and the outcome of the service g becomes $g(f(x))$. Hence, we do not classify the service itself as a resource, rather, the outcome produced by the service is again, a resource. Since the act of providing a resource is itself a service, for the sake of simplicity we will occasionally say *service discovery* where we mean *discovery of services and resources*.

In this survey, we considered existing service discovery approaches in large-scale networks. Most of the industrial service discovery approaches like (Jini, SLP, UPnP etc.) use hardware devices like printer, fax machine, etc. as networked services. On the other hand, the majority of WebService discovery approaches consider a software component, performing a specific functionality, as a service. Our definition of "service" covers both views.

- Service description: The set of descriptive attributes of a particular service.
- Service advertisement: Both the publication of information about a service, and the published information itself, are referred to as a service advertisement
- Soft-state advertisement: When a lifetime is associated to a service, the advertisement is said to be soft-state. If the advertisement is not renewed before the expiration of the lifetime, then the service is assumed no longer available.
- Hard-state advertisement: In contrast to the soft-state advertisement, a hard-state advertisement assumes an infinite service lifetime. A service is assumed to be available until its unavailability is explicitly announced.

Description of Criteria

The primary objective of our investigation is to establish the foundation for building a distributed large-scale, multi-domain discovery system capable of facilitating open service discovery across heterogeneous software/hardware infrastructures, and of scaling to hundreds of thousands of users. To this end, we structure our analysis and comparison of the key discovery approaches, based on a set of evaluation criteria that we believe are critical to the construction of such an open and distributed discovery system. To the best of our knowledge, this is the first comparison in literature undertaken with a strong focus on designing large-scale service-discovery systems across service discovery domains, administrative domains and network boundaries. Of the few existing works on comparing discovery systems [9–11], we find their criteria applicable to limited context (e.g. ad hoc networks, LAN). Research in the area of designing discovery systems has been prolific in the past years, and has resulted in many existing discovery systems designed for diverse application environments (e.g. WLAN, home networks, grid services, etc.). Some of our criteria are chosen to categorize these systems, such as by architecture and by scale. However, our criteria largely focus on evaluating whether each discovery system is suitable for an large scale multi-domain setting. We consider the following aspects in particular: effectiveness, fault tolerance, performance, security, and platform- and network-dependence. These criteria are generated from our investigation into large-scale and multidomain networks (Fig. 1). In addition, we list the standardization efforts and implementation status of these systems as an indicator of their maturity and acceptance.

The following subsections provide a brief description of each criterion and its importance in our context. Due to the limitation of space, our latter evaluation of existing discovery systems selects only those systems that best illustrate a certain technique or class of approaches.

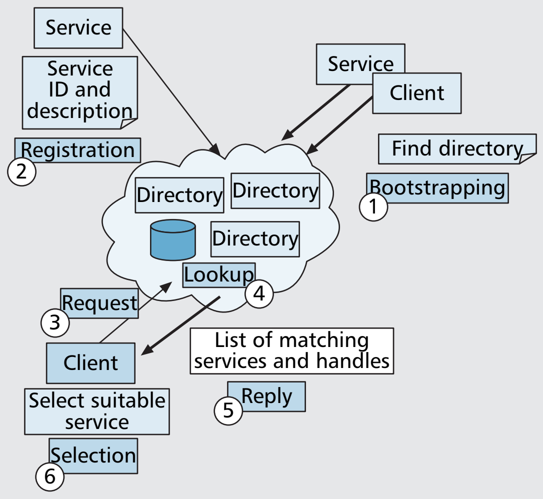
Architecture

This criterion refers mainly to how information is distributed, and therefore accessed, in the discovery infrastructure. The architecture of discovery systems ranges from *centralized* to fully *decentralized*. It is a key design criterion influencing how many requests, in a given time interval, the system can handle and how efficient it can execute these requests. The nature of the Internet favors distributed architectures, however the very degree of distribution is much dependent on the way information is to be stored, and how it is accessed.

Effectiveness

An important criterion in examining any system is how effectively it is able to perform its tasks. In our case, we look at how well the system can discover services; specifically, we look at the correctness and completeness of discovery. The terminology used in this context varies across literature. Information retrieval literature uses *precision* and *recal*l to describe the effectiveness of data retrieval operations. We prefer to use the terms well known in peer-to-peer literature: *completeness* and *correctness*. Completeness is the ability of a service-discovery process to return all matching service instances registered in a service directory. If there are multiple directories (or service registries) accessible to the system, every matching service registered in the directories must be retrieved for the discovery process to be complete. The other criterion, correctness, denotes the validity of the results of service discovery: when a service is discovered, how closely does it match the user's request?

In a multi-domain environment that hosts thousands of service instances, both correctness and completeness of discovery are more important than in a small, restricted environment. Completeness of service discovery requires the system to retrieve relevant results across domain boundaries and across potentially-large network distances. Complementarily, a high degree of correctness is necessary so that these relevant results do not get lost among a flood of a possibly huge number of irrelevant results. In an ideal situation, the correctness measures of a service discovery system would ensure that the returned results closely match the intent of the query, despite the heterogeneities that may exist in the information repre-



Issues (discovery process)

Bootstrapping	- Communication model - Involved entities
Service advertisement	- Communication model - Directory/advertised information - Advertisement persistence
Querying	- Query expressiveness - Communication model - Routing model - Lookup method - Query result
Service handle retrieval	Type of the service handle

General issues (discovery process)

Architecture	Distribution of the directory information
Effectiveness	- Correctness - Completeness
Security	- Authentication - Authorization - Privacy - Data integrity
Fault-tolerance	- Points of failure - Recovery method
Performance	- Communication overhead - Load balancing
Scalability	Ability to handle large number of clients/services in large networks
Platform/network independency	- Platform dependency - Network dependency
Interoperability	- Ability to interoperate with other systems - Non-interoperation reasons
Standardization	Standards and existing implementations

Figure 1. *Service discovery systems: concepts and issues.*

sentation and syntax of each service-discovery domain. While completeness depends mostly on the search mechanism, correctness is largely achieved through the expressiveness of service description and query languages.

Fault Tolerance

Fault tolerance deals with the behavior and robustness of the discovery process under erroneous conditions. Since the discovery system is a support function to service infrastructures, its failure may hinder the entire service provisioning process. We consider two aspects of fault tolerance: how does the system behave under erroneous conditions and how well does the system recover from errors. Robustness of the discovery system in terms of these two aspects is crucial in a multidomain environment with hundreds of thousands of users and services, as inevitably such system will encounter operational errors. Its recovery (or failure to recover) will have significant impact to users/systems over a large scale. In each discovery approach, we examine the resistance to failure in the system and its recovery mechanisms.

Performance

Because the discovery process is a managerial task, the amount of communication overhead involved (i.e. network resources devoted to managerial tasks) is an important design consideration. Under extreme cases, the communication overhead of discovery can reduce the effective use of network resources to an unacceptable level. In a large-scale service infrastructure, the discovery service should be able to cope with a large amount of consumer requests. Hence, we need to consider the challenges of optimizing the communication overhead and providing load balancing. It would be ideal to evaluate and compare all of the examined discovery systems based on a quantitative set of performance measures or benchmarks. Unfortunately, there are no comprehensive benchmarks or evaluation measurements of service discovery approaches; existing works do not cover a large enough set of discovery systems and are aimed at very limited application scenarios. It is difficult to define a set of objective performance measures that are universally applicable to all discovery systems, and even more difficult to quantitatively evaluate

	Salutation	SDS	SLP	Jini	INS	UPnP	WS/GS	JXTA	INS/Twine	Splendor	P2P
Architecture		√							√		√
Effectiveness									√		√
Fault-tolerance		√			√						√
Performance		√			√				√		
Security		√								√	
Platform and network independence	√			√			√	√			
Scalability									√		√
Interoperability							√	√			
Standardization	√		√	√		√	√	√			

Table 1. *Suitability of representative approaches against the selected set of criteria.*

them in a largescale settings. In this article we focus our attention instead on providing good set of qualitative evaluations of the systems, and to provide objective descriptions that could indicate their performance from an architectural point of view. To this end, we examine the amount of communication overhead involved in the discovery process, which affects both the utilization of underlying network resources and the latency of query response. We also look at the treatment of load balancing.

Our discussion (and specifically in Tables 5, 6 and 7) uses the qualitative terms of *high*, *moderate*, and low to express performance measures in terms of communication overhead. High communication overhead implies that extensive communication cost is involved, for example when broadcasting query messages and responses. Moderate overhead means a moderate use of network resources, for example, multicasting query messages to small groups. Finally, low overhead is generated by operations that are very resource conscious, for example, point to point communication between users/services and a directory.

Security

Security issues in a single administrative domain can be addressed using intra-domain security policy mechanisms. However, security concerns in large-scale and multi-domain environment are both more challenging and more critical, since the system is exposed to a much larger population of potential attackers, and security policies are harder to enforce. How can user identity be verified and its access controlled? How can we ensure crucial discovery or service data are not tempered with? How can user be assured that its sensitive data are protected from prying eyes and its identity hidden from other unauthorized users? These issues are significantly more challenging and critical in a large-scale and multi-domain environment.

In this respect, we look at how each discovery system handles the following security aspects: user authentication/ authorization, data integrity, and privacy of sensitive data. For each discovery system, we examine how it copes with these aspects and where it lacks security measures.

Scalability

Many of the existing discovery approaches are designed with a particular target infrastructure size in mind, regarding network size and number of consumers and services supported. Our consideration of scale pertains to "large size" networks and global service infrastructures with hundreds of thousands of users and end systems. This criterion examines the targeted infrastructure size of each systems and allows us to evaluate the ability of discovery systems to cope with large scale networks.

Platform and Network Independence

A strong dependence on specific technologies may prevent a discovery system from being deployed in multi-domain environments which are likely to use different platforms and networking technologies. Looking at the dependencies of discovery systems on particular software platforms or network technologies allow us to evaluate their ability to operate in a multi-domain environment.

Interoperability with Other Discovery Services

For a discovery system to operate over existing infrastructure across multiple domains, it must interoperate with other (potentially heterogeneous) discovery systems. We examine the ability of each discovery approach to interoperate with others, and identify the reasons that may allow or prevent their interoperation.

Standardization and Existing Implementation

This criterion gives a good indication of the maintainability and future outlook of a discovery service. Standardization and the presence of existing implementation may ease the inclusion of a particular discovery service in a large-scale service infrastructure.

Technology Overview

This section provides an overview of the architecture design and the discovery process proposed by each approach. We view service discovery as a four-step process (Fig. 1);

- *Bootstrapping*, where clients and services attempt to initiate the discovery process via establishing the first point of contact within the system

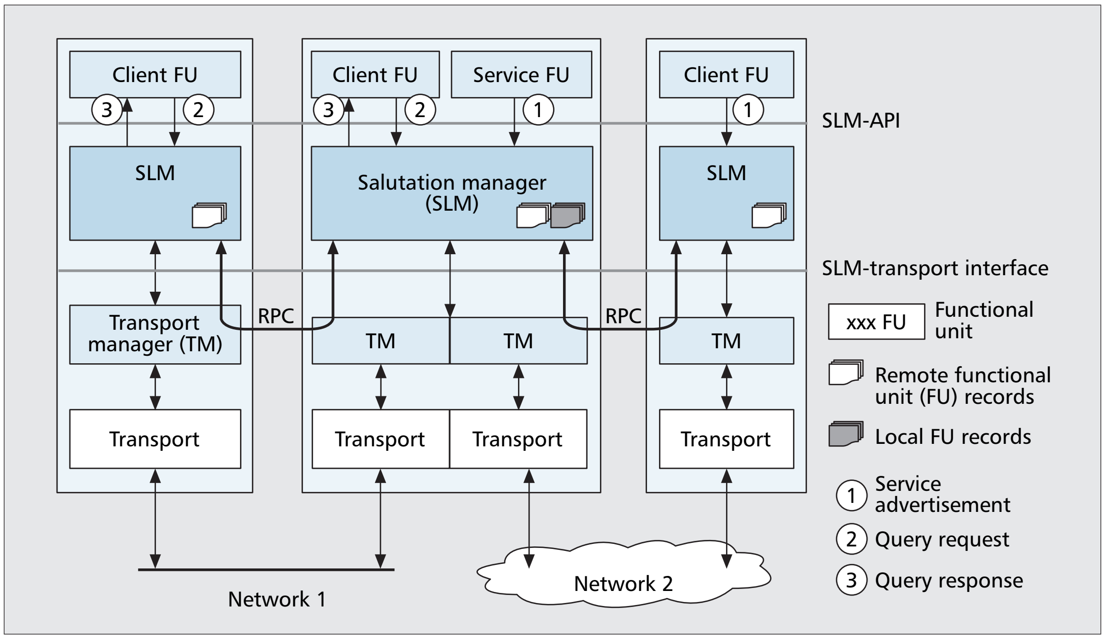


Figure 2. *Salutation architecture.*

- *Service advertisement*, where a service provider publishes information about the provided services
- *Querying*, where a client looks for a desired service
- *Service handle retrieval*, the final step in the discovery mechanism, where a client receives the means to access the requested service

Note that some of these steps may be omitted in various discovery approaches. This Section will describe how these steps are implemented. For each described step of the discovery process, we will concentrate on some specific features or issues as listed in Fig. 1, e.g. at the querying step, we will discuss the expressiveness of the request, the communication model, the routing model, the lookup method and the query result. We will hence highlight the major characteristics of each approach.

As discussed earlier, the chosen criteria are the most important ones for large-scale, cross-domain service discovery. In this section we examine the discovery approaches that provide a suitable solution for one or more criteria presented earlier. We have chosen only a minimal set of approaches to keep the survey concise, while representing the state of the art in the field. Table 1 presents the suitability of the selected set of service discovery approaches against the selected set of criteria. A check-mark in row R and column C indicates that approach C is a good fit for criterion R. There exists a number of discovery approaches other than the ones discussed in this survey. These approaches either focus on the issues that are out of the scope of this survey, or they do not provide better solution than the selected approaches. For example, context-aware service discovery has been considered in [12] and [13]. Many service discovery systems assume single hop (e.g. Bluetooth [6] and Deap-space [7]) or multi-hop (e.g. Konark [8]) ad hoc wireless networks. For our envisioned wide-area, multi-domain service discovery, neither context-awareness nor ad hoc environment support are of significant importance. To keep the survey focused we have not considered these approaches. In essence, we have selected the most prominent service discovery approaches, from industry and academia, that we consider to be potential candidate for our envisioned large scale, multi-domain service discovery system.

SALUTATION

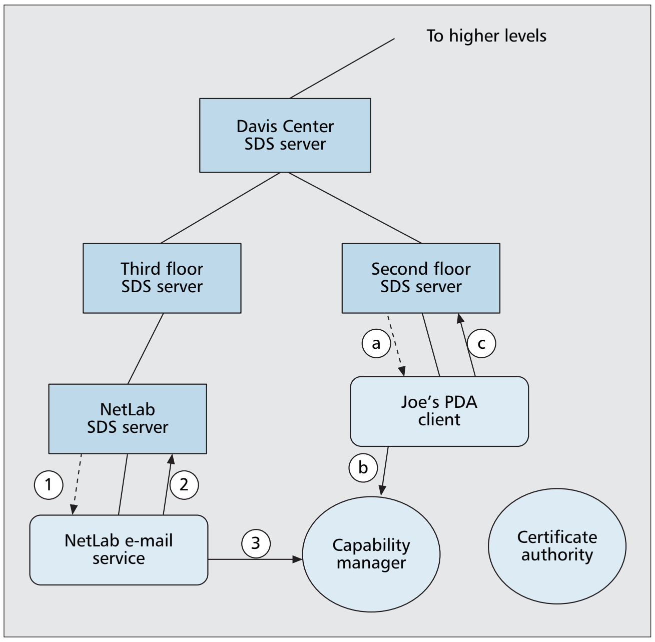
Salutation [14, 15], a product of the Salutation Consortium which was founded in 1995, is a service discovery system that targets heterogeneous environments of widespread connectivity and mobility. From an architectural point of view, Salutation is a decentralized system based on a core entity called Salutation Manager (SLM). Each Salutation-enabled device implements a local SLM. An SLM stores and maintains information about local services, acting as a service broker for local clients. It collaborates with remote SLMs in order to exchange the capabilities of registered services. SLMs discover each other and communicate even if they act on top of different transport media. These functionalities are ensured by transport-dependent modules called Transport Managers, as shown in Fig. 2.

In order to enable communication among SLMs, each SLM maintains the list of identifiers of remote SLMs, while the Transport Manager maintains the association between the identifiers of remote SLMs and their corresponding transport addresses. Each SLM is identified by a Universal Unique Identifier (UUID) [16]. If the Transport Manager has no a priori knowledge about existing remote SLMs, it can discover them either by broadcasting a discovery request, to which remote SLMs may respond, or by inquiring a central repository (if it exists) that would maintain information about Salutation enabled devices.

Salutation Managers can also be involved in managing sessions between clients and remote services. Communications between SLMs use Remote Procedure Call (RPC). The exact protocol used is the Sun Microsystems Open Networking Computing Remote Procedure Call version 2 [17].

Bootstrapping — Salutation defines the SLM-API (Fig. 2), an application programming interface provided by the Salutation Manager to develop Salutation applications.
On startup, Salutation clients and services invoke the SLMAPI to associate with the local or the nearest available SLM. The association process consists in registering the *functional units* corresponding to the client or service. In Salutation architecture, functional units are used to define all service functions, as well as certain client functions.

When there is no local SLM, then the SLM-API of a



■ **Figure 3.** *SDS architecture. (1),(a) SDS server announcements on multicast channel; (2) Secured advertisement by service; (3) Announcement of access control list (<principal, service class> pairs) by service; (b) client learns its capabilities form CM; (c) client submits query and its capabilities to SDS server CM acts as a mediator for propagating capabilities from services to clients.*

remote SLM is invoked through RPC.

Service Advertisement — A service advertises itself by registering with the SLM a *description record* using the hard-state model. This record follows the ASN.1 (Abstract Syntax Notation One) from OSI as per the notation/encoding scheme and is composed of one or more functional units. Each functional unit represents a description of a functionality of the service and contains a handle to that functionality.

Querying — The querying process is handled by the SLM with which a client is associated; the client may first ask its SLM to find out which SLMs provide a desired service, i.e. the service that matches the description record provided by the client, and finally select one and ask its SLM to retrieve the description record from that selected SLM.

SLMs have the possibility to cache remote service descriptions. In that way, an SLM can respond to its associated clients' queries with the cached descriptions. However, since service registration is hard-state, and there is no explicit way to refresh remote caches, the consistency of remote caches and then the accuracy of a response based on the local SLM's cache is not guaranteed. Instead of lease-based service registrations, SLMs can be asked by clients to periodically check the availability of service functional units.

Service Handle Retrieval — Clients receive handles to functional units in response to their queries. Salutation Managers may be involved in managing sessions between clients and remote services.

The interoperation of heterogeneous devices is the primary concern of the Salutation framework. Beyond, abstracting the transports through the use of Transport Managers, Salutation also provides a lighter version, called Salutation-Lite [18], that targets small devices with restricted capabilities. These considerations are important in a multi-domain environment. However, conceptually, Salutation is not designed for large networks, and its deployment is currently restricted to office automation products like fax machines, printers, copiers, and scanners.

SECURE SERVICE DISCOVERY SERVICE (SDS)

The Secure Service Discovery Service [19] (sometimes abbreviated as SSDS) was designed and developed by the Computer Science Division, University of California, Berkeley, in 1999. SDS is a research project and aims to achieve security and wide area support.

Besides services and clients, the SDS architecture is composed of the following three components:

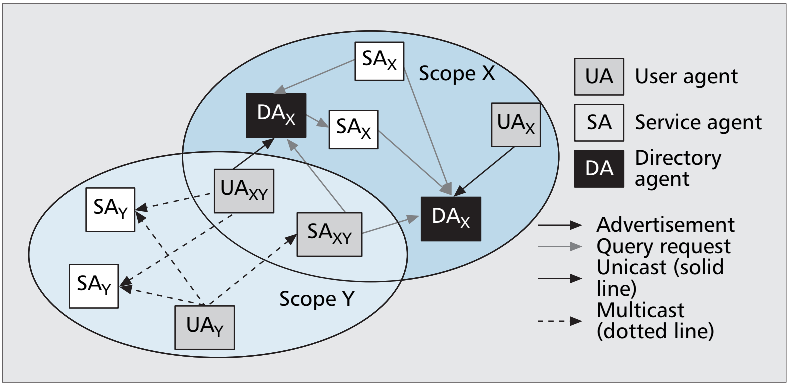
- **SDS servers** are directory agents, arranged into a treelike hierarchical structure. Each SDS server gathers service descriptions from services or child SDS servers in the form of Bloom filters [20]; aggregates these advertisements using bit-wise OR; and propagates the aggregated Bloom filters to its parent. SDS allows multiple hierarchies to co-exist. A SDS server can simultaneously participate in multiple hierarchies by maintaining multiple sets of parent-child pointers. These hierarchies can be based on administrative domains, network topology, geographic location etc.
- **Certificate authority** is a trusted central component responsible for verifying the digital signatures used to establish the identities of different components in the SDS architecture.
- **Capability manager** (CM) uses capabilities (access rights) for controlling the visibility of a class of service to a principal (set of users, e.g. students of physics department). Capabilities are signed messages, indicating that a particular class of service descriptions can be discovered by the users from a particular principal.

In SDS architecture all of these three components are trusted entities which use authentication and encryption for security measures. Communications between different entities in the system are authenticated, whenever necessary. SDS relies on soft state technique for service advertisements and capabilities. Service advertisements are encrypted using a hybrid public key/symmetric key mechanism. Message header contains a symmetric key, and is encrypted with the public key of the receiving SDS server. Message body, on the other hand, is encrypted with the symmetric key embedded in the message header. This allows faster decryption of the message body using symmetric key technique, rather than using slower public/private key decryption for the whole message. SDS uses XML documents for describing both service descriptions and queries.

Bootstrapping — SDS servers periodically announce their existence on a globally known multicast channel. Such announcements include information like:
- The list of administrative or network domains the SDS server is responsible for
- The multicast address to use for service advertisement
- The desired service advertisement rate
- The contact information for the Certificate Authority
- The contact information for the Capability Manager

Services and clients have to continuously listen on the SDS multicast channel to learn these information and eventually to join the system.

Service Advertisement — A service has to continuously listen to the SDS multicast channel to find the appropriate SDS server for its advertisements (Fig. 3). Finding the correct SDS server is not a one-time task because new SDS servers can



■ **Figure 4.** *SLP architecture.*

appear or existing SDS servers may crash.

After determining the appropriate SDS server, the service has to multicast its service description using the hybrid public/symmetric key technique, using the public key of the selected SDS server for encryption. The service has to contact the capability manager and inform the list of principals allowed to discover the service.

Querying — A client communicates with a SDS server using Authenticated RMI (remote method invocation) and submits a query (in XML) along with its access rights. The SDS server searches its local database and returns all the service descriptions matching the query and the client's capabilities. The SDS server forwards the query to its parent, in case no match is found in the local database.

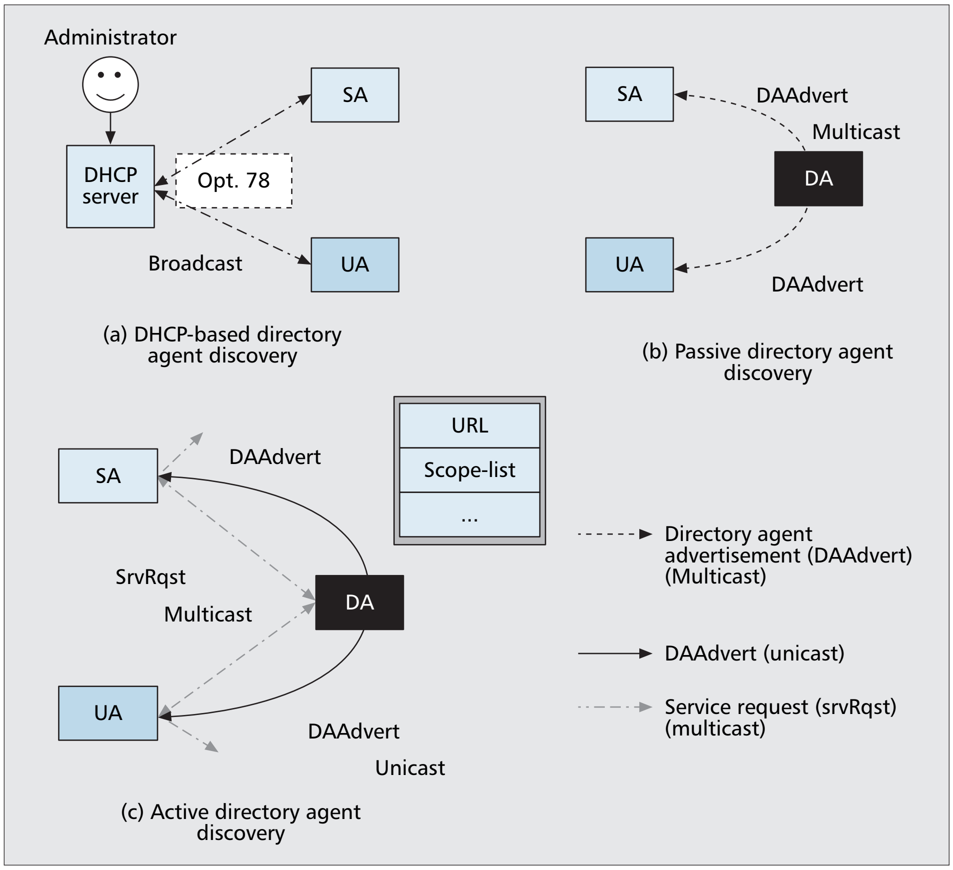
Service Handle Retrieval — As a result of a successful discovery process the client obtains a set of XML documents representing the descriptions of the matching services. A typical service description includes the location of the service, expiry time of the advertisement, Java RMI address of the service, and other functional and non-functional attributes of the service.

SLP

Service Location Protocol (SLP) [21, 22] was introduced by the IETF SVRLOC Working Group and initiated in 1997. SLPv2 is the last version of SLP and was standardized in 1999.

SLP was conceived as a lightweight, open and scalable protocol for service discovery, and is generally intended to function within IP networks under single administrative control, i.e. enterprise-like networks. It follows a *hybrid* architecture, where decentralized and centralized, directory-based and non-directory-based architectures are possible and can coexist. The directory-based architecture relies on a Directory Agent (DA) that takes care of registering services and responding to queries. Two other entities also participate in the discovery process; a User Agent (UA) that acts on the consumer's behalf, and a Service Agent (SA) that acts on the service provider's behalf.

SLP introduces the notion of *scope*. The primary use of scopes is to provide the ability to create administrative groupings of services. A consumer seeking services is configured to use one or more scopes; it will only be able to discover the services that belong to its scopes, as shown in Fig. 4. By con-


Administrator
DHCP server
Opt. 78
SA
UA
Broadcast
(a) DHCP-based directory agent discovery
SA
DAAdvert
Multicast
DA
UA
DAAdvert
(b) Passive directory agent discovery
URL
Scope-list
...
SA
DAAdvert
SrvRqst
Multicast
DA
UA
DAAdvert
Unicast
(c) Active directory agent discovery
Directory agent advertisement (DAAdvert) (Multicast)
DAAdvert (unicast)
Service request (srvRqst) (multicast)

■ **Figure 5.** *SLP directory agent discovery.*

figuring UAs and SAs with scopes, administrators may provision services.

Bootstrapping — When bootstrapping, UAs and SAs may or may not have pre-configured scopes. In the latter case, a UA or SA can get its scope list in response to a DHCP request, when the DHCP SLP Service Scope Option [23] is used. On the other hand, when a UA is configured with a "NO SCOPE" value, the UA is assumed to obtain its own selection of scopes. This is similar to Samba's user-selected workgroups.

Once configured with scopes, the UAs and SAs need to contact DAs. When access to DA is not pre-configured, UAs and SAs can use the following methods of locating a DA: *DHCP-based configuration*, *DA active discovery*, and *DA passive discovery*. The DHCP-based configuration consists in taking advantage of the DHCP SLP Directory Agent Option (referred to as option number 78) [23] to retrieve the list of IP addresses for DAs within specific scopes. The DA active discovery is performed by initiating a multicast DA request (or broadcast if multicast is not supported); UAs and DAs may get unicast responses from DAs within their scopes. Finally, DA passive discovery consists of listening to the DA advertisement messages multicasted (or broadcasted) over the network. Figure 5 shows these three DA discovery processes.

Service Advertisement — If a DA is associated to a scope, then all SAs belonging to that scope must register their services with the associated DA as per a soft-state model. A registration contains the type of the service, its URL and a set of descriptive attribute-values pairs. Service types and attributes may be vendor-specific or assigned by a naming authority, which is the Internet Assigned Numbers Authority (IANA) by default [24, 25].

Querying — When a UA is configured with a scope that contains a DA, then the queries must be unicasted to that DA. Queries may contain the type of the desired service, and a number of predicates over its descriptive attributes. These predicates follow the LDAPv3 search filter format.

The following is a generic example of an SLP request. Assuming that a user u needs to access a service srv, and that it is associated with two scopes Scp and DEFAULT, the query looks like the following:

```
<t>=service:srv <s>=Scp,DEFAULT <p>=(user=u)
```

Here $< t >$ indicates the requested service type, $< s >$

gives the $< scope - list >$ and $< p >$ is the predicate string, i.e. the list of predicates over service attributes.

Queries are matched against registrations. However, if no DA is associated with a scope, then the UAs and SAs belonging to that scope interact directly; UAs multicast queries according to the SLP-specific *multicast convergence algorithm* [21], and receive back responses from SAs.

Service Handle Retrieval — The URLs of matching services are returned to the consumer as access points to these services.

In addition to the flexibility and lightweight of the architecture, SLP also presents interesting features in terms of performance and security, as we will see later. Being an IETF standard, SLP is relatively widely implemented and commercialized. In addition to several office and networking devices, several platforms support SLP, including Sun, Caldera, Novel, and Apple.

Jini

Jini [26] was introduced by Sun Microsystems in 1998. The Jini Community was initially established in January 1999 with the release of the first version of the Jini Technology Starter Kit from Sun Microsystems. The last version of the Jini Technology Starter Kit has been released in February 2004.

Jini aims to provide a platform for dynamically creating networked components, applications, and services. It typically targets enterprise environments.

Similarly to SLP, Jini allows different architectures to be implemented. It provides a Lookup Service (LUS) that acts as a Directory Agent. Again similarly to SLP, all Jini entities are "scoped"; they are grouped into federations called "djinns."

Bootstrapping — On bootstrapping, clients and service providers try to discover the LUS associated to the "djinn" they belong to. This may be done either through the *multicast announcement protocol*, i.e. by listening to the multicasted LUS advertisements, or through the *multicast request protocol*, i.e. by sending multicast requests for Lookup Services.

Service Advertisement — Once a LUS is found, services register with it. Registrations are lease-based. A registered record is an instance of the *serviceItem* Java class shown below:

```
public class ServiceItem implements Serializ-
able {
public ServiceItem(
ServiceID serviceID,
Object service,
Entry[] attributeSets)
{...}
public ServiceID serviceID;
public Object service;
public Entry[] attributeSets;
}
```

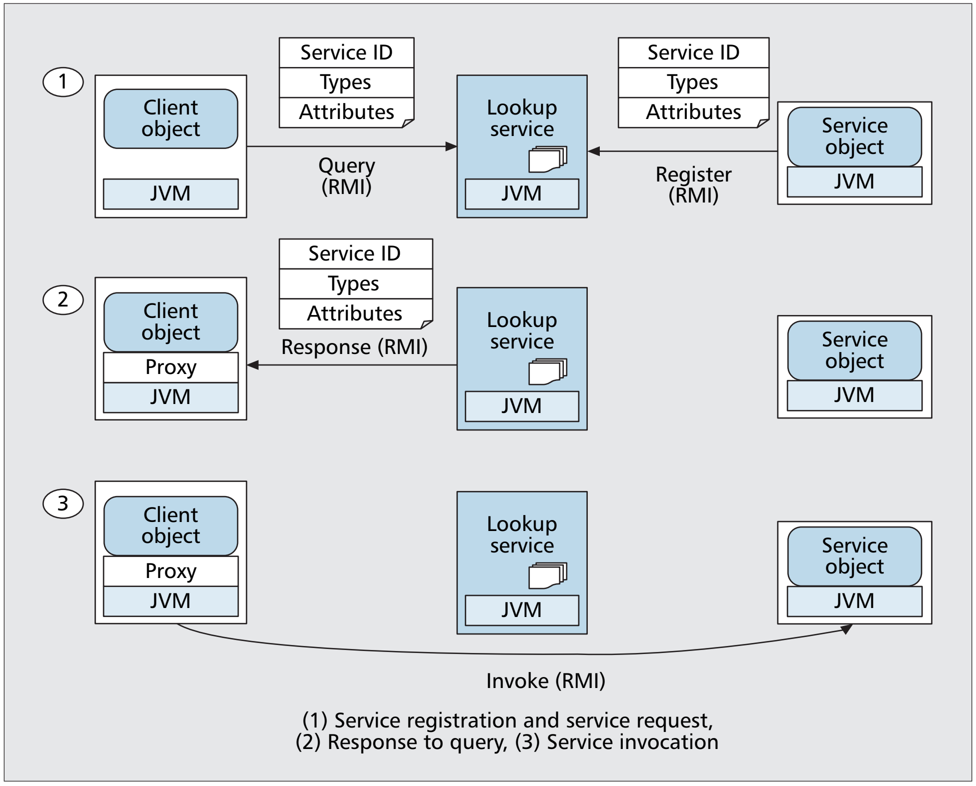
A service registration contains the service assigned UUID (*serviceID*), a proxy-object (*service*) and a set of attributevalue pairs describing the service (*attributeSets*).

Querying — When a LUS is found, client's queries are sent to that LUS using Remote Method Invocation (RMI). A query object is an instance of the following *ServiceTemplate* Java class:

```
public class ServiceTemplate
implements Serializable {
public ServiceTemplate(
ServiceID serviceID,
Class[] serviceTypes,
Entry[] attributeSetTemplates)
{...}
public ServiceID serviceID;
public Class[] serviceTypes;
public Entry[] attributeSetTemplates;
}
```

where *serviceID* refers to the UUID of the requested service. The *serviceTypes* include the list of Java classes the service may be an instance of, and *attributeSetTemplates* is a set of attribute-value pairs describing the requested service.

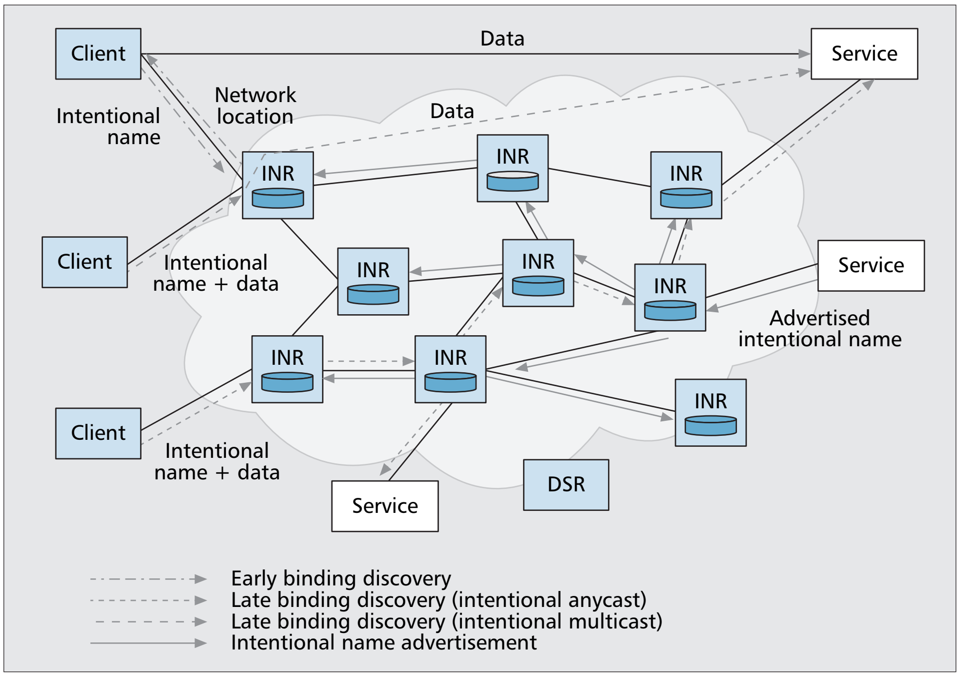
The LUS matches the query against the registered records. A registered record (*item*) matches a query object (*tmpl*) if *item.serviceID* equals *tmpl.serviceID* (or if *tmpl.serviceID* is null), *item.service* is an instance of every type in *tmpl.serviceTypes*, and *item.attributeSets* contains at least one matching



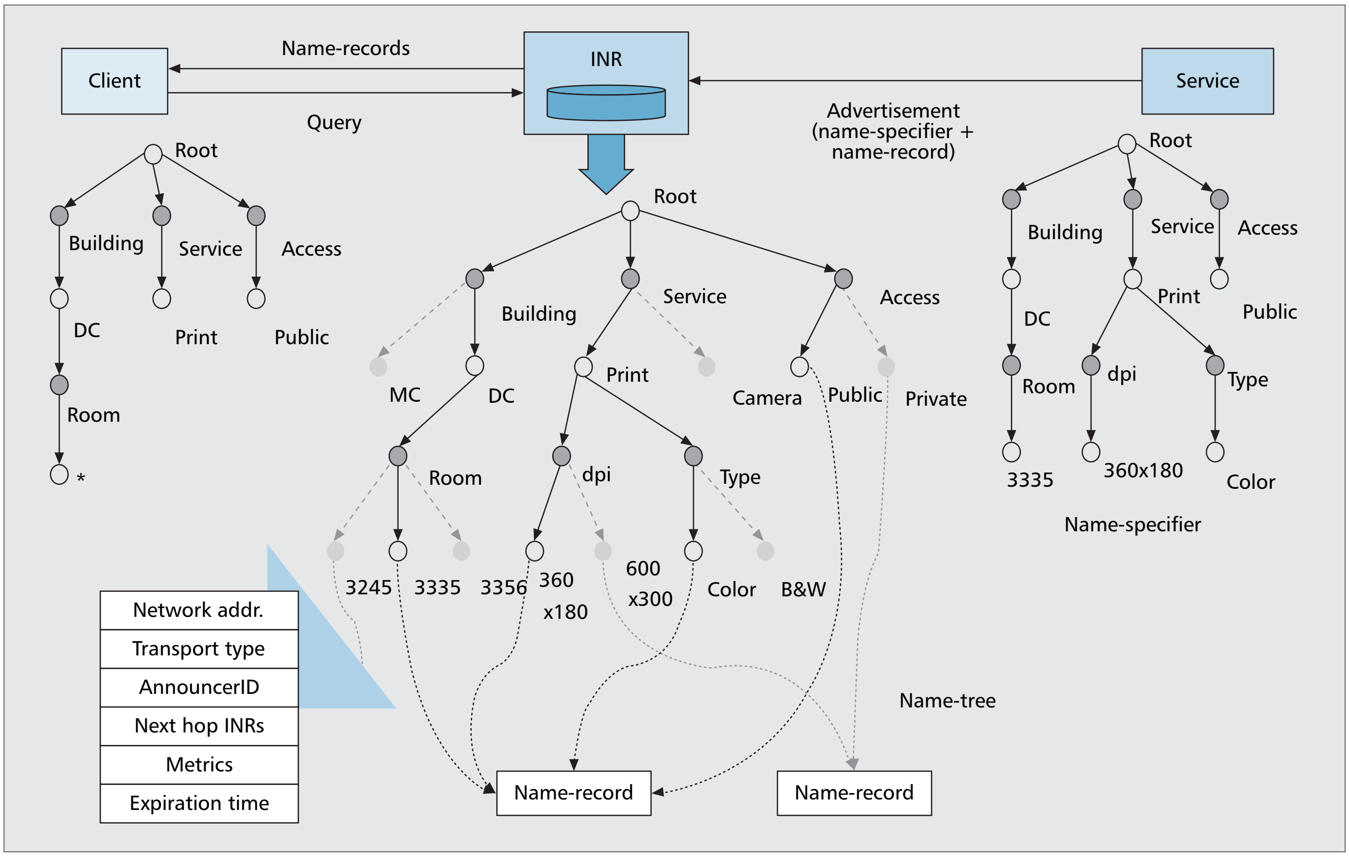
■ **Figure 6.** *Jini discovery process.*

entry for each entry template in *tmpl.attributeSetTemplates*. This pattern matching ensures the accuracy and the completeness of the query result in the limit of the number of responses the client may wish to receive, and of the scope of the associated "djinn." A filter may also be implemented in the client side in order to make a selection over the received responses.

It is worth noting that if no LUS can be reached, a client may act in its place, provided that LUS functionalities are implemented on the client side. In this case, the client sends a LUS multicast announcement so that service providers can register with it. The client can then select from these advertisements the ones it is interested in. This mechanism is commonly known as peer *lookup*. It should be noted that, the client uses peer lookup only to discover services; it neither starts functioning as a LUS, nor responds to service discovery requests from other clients. In peer lookup mode, communication between services and clients starts over multicast com-



■ **Figure 7.** *INS architecture and discovery modes.*



■ **Figure 8.** *INS discovery mechanism.*

munication channel similar to that of SLP DA-less mode.

Service Handle Retrieval — The client receives one or more than one instances of the ServiceItem class, each referring to a service matching the client's request. When the response contains a proxy-object, this one is used as a service handle to access the service through RMI. The discovery process and the access to the service are shown in Fig. 6.

The use of Java technology, and hence Java Virtual Machines, abstracts the underlying transports and platforms. Jini also takes advantage from major Java community achievements such as RMI which provides code mobility, and lately, the Davis project [27] a Java-based security framework which addresses major security issues (authentication, authorization, integrity and confidentiality). In addition to these features, Jini provides an event notification mechanism that allows objects to be triggered.

INS

Intentional Naming System (INS) [28] was introduced by the MIT Laboratory for Computer Science in 1999. It provides protocols for service naming and discovery in dynamically changing and mobile networks.

INS discovery system is performed using an application level overlay network of participating directory agents called Intentional Name Resolvers (INRs) (Fig. 7). The directory information is distributed among the INRs.

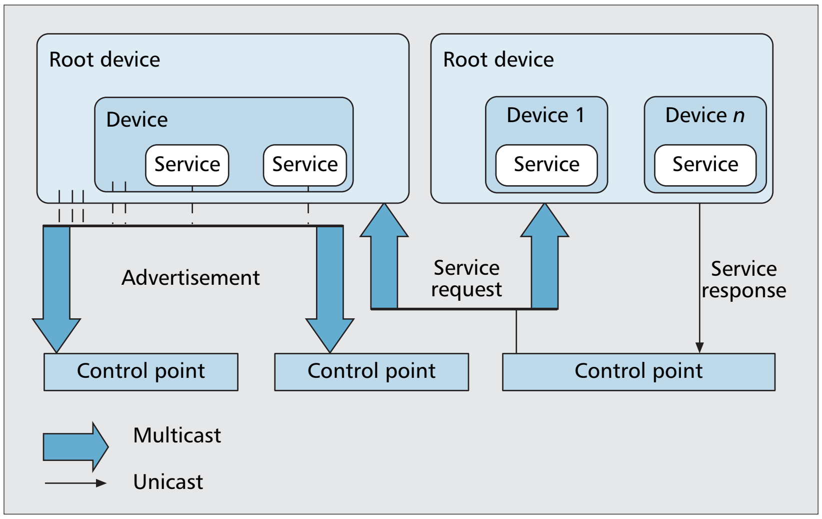
For bootstrapping INRs, a centralized agent called Domain Space Resolver (DSR) is used. This agent keeps track of all active INRs in the system and provides this list to each joining INR. Upon joining the network, an INR selects another INR as its neighbor based on the shortest round-trip time required to ping each active INR in the system. This mechanism enables the resolver network to maintain a spanning tree topology. The INRs can join or leave the system dynamically and self-organize into a spanning tree. The spanning tree topology is used for application-level multicasting and for propagating advertisement information.

Bootstrapping — The bootstrapping of consumers and service providers is not considered; they are assumed to know the IP address and the port number on which INRs are listening.

Service Advertisement — On startup, service providers register their services with an INR, according to a soft-state model, by providing both a *name-specifier* and a *name-record*. The former consists of a descriptive name that is composed of attribute-value pairs with a hierarchical tree-like relationship. The latter contains complementary information about the service (like the IP address and port number of the service, the requested transport protocol, the service lifetime, etc.) that may be returned to the clients in response to their queries. For better responsiveness to client queries, service advertisements are replicated in each INR. To keep each replica consistent, INRs exchange information in regular intervals. Triggered updates may also be implemented.

All the name-specifiers known to an INR are stored in a tree-like data structure called a *name-tree*, which is effectively an attribute-wise union of all the name-specifiers; for example if two name-specifiers have a common attribute x at the same level with values A and B respectively then, in the name-tree A and B, are placed as children of x. The leaf nodes in a name-tree point to the name-records that were advertised with the name-specifiers. An example of name-tree is shown in Fig. 8.

Querying — The same name-specifier scheme is used in queries to describe the requested service. Clients also specify



■ **Figure 9.** *UPnP architecture and discovery process.*

in their queries the expected behavior of the consulted INR; they expect either to receive the matching service namerecords (*early binding* discovery mode), or the matching service to be contacted on their behalf (*late binding* mode). In this last mode, the client's query also contains the data that will be forwarded to the service. The INR may contact either a selected matching service based on client's metrics, issuing what is called *intentional anycast*, or may forward client's data to all matching services, by a process called intentional multicast. Figure 7 illustrates INS discovery processes.

Upon receiving a query, an INR browses the entire stored name-tree in order to look for the matching name-records. The complexity of the lookup process is $O(n_a{}^{d/2})$ [29] where na is the total number of attributes and d is the depth of the name-tree. Hence, the lookup process does not scale well with increased number of name-specifiers and attributes. In addition, the storage space needed to store the name-tree increases rapidly with the number of non-identical name-specifiers and attributes.

Service Handle Retrieval — When early binding is used, the client receives in response to its query one or several namerecords corresponding to instances of the requested service. A name-record contains the following information:

- *Network address* of the service, i.e. IP address and port number.
- *Transport-type* (e.g., TCP, HTTP, RTP etc.) supported by the service. This is used by the client to implement early binding.
- *AnnouncerID*, a unique identifier of a service, constructed from host IP address and the service creation time. AnnouncerID is used to distinguish between services advertising the same name-specifier and residing in the same host.
- A set of routes to next-hop INRs and the metrics for each route (e.g. hop-count, transmission delay etc.). This information is used for intentional multicasting.
- Service provider metrics (e.g. average load), used for service selection in anycast mode of discovery.
- Expiration time of the name-record.

INS presents a unique discovery architecture and an interesting service naming scheme. However, it is a relatively heavy consumer of storage-space, computation and bandwidth, which raises a scalability issue. INS/Twine [29] has been designed as an enhancement for INS especially in terms of performance and scalability.

UPnP

UPnP [30], publicized primarily by Microsoft, is maintained by the UPnP Forum which was founded in 1999. The first version of the UPnP device architecture was released in 2000. A recent update, dated in 2003, is still at proposal status. UPnP technology provides a decentralized, open networking architecture that uses TCP/IP and Web technologies (like HTTP and the eXtensible Markup Language (XML) [31]) to enable *seamless proximity networking* [30] in managed and unmanaged small networks. UPnP uses the Simple Service Discovery Protocol (SSDP) [32] to discover and to announce services. This protocol was presented to the IETF by members of the UPnP Forum. However it is still a draft and has not been standardized yet.

UPnP is a fully decentralized peer-to-peer approach. UPnP devices announce themselves and their supported services on the network. Control Points, which act on the consumer's behalf, catch the interesting announcements and can also initiate queries based on consumer's needs.

Centralized	Registry		Authoritative, centrally controlled store of service descriptions, e.g. UDDI registry [36]
	Index		Non-authoritative, centralized repository of references to service providers; see [1] for details.
Decentralized	Federation		Publicly available UDDI nodes collaborates to form a federation and acts together as a huge virtual UDDI registry [38].
	P2P-based	Semantic	In [39] peers are arranged into hypercube and ontology is being used to facilitate efficient and semantically-enabled discovery. Another approach [40] presents agent-based solution and uses DAML representation for ontology.
		Non-semantic	Both [41] and [42] uses Chord [43] overlay for indexing and locating service information. These approaches vary in the method of mapping service descriptions into indexes. [41] extracts keywords from service descriptions and uses MD5 hashing. [42] uses Hilbert Space Filling Curves for mapping similar service descriptions to nearby nodes in Chord ring.

■ Table 2. *A taxonomy of web service discovery architectures.*

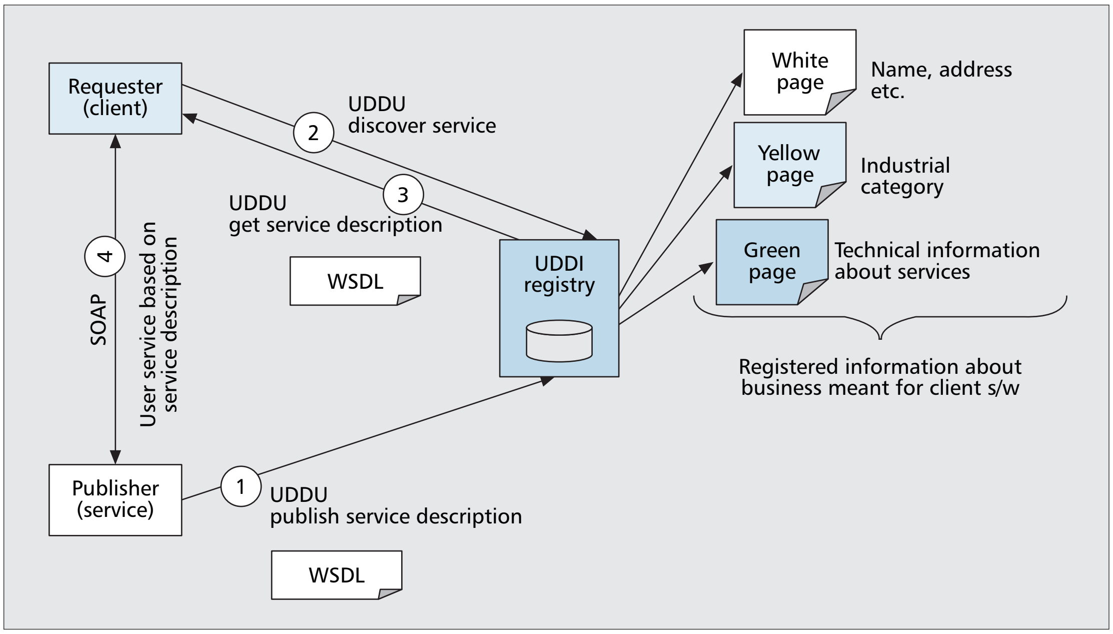

White page
Name, address etc.
Requester (client)
UDDU discover service
2
Yellow page
Industrial category
3
UDDU get service description
Green page
Technical information about services
4
SOAP
User service based on service description
WSDL
UDDI registry
Registered information about business meant for client s/w
Publisher (service)
1
UDDU publish service description
WSDL

Figure 10. *UDDI discovery mechanism.*

Bootstrapping — In contrast with other discovery systems, UPnP acts on a lower level. It addresses for example the problem of automatic assignment of IP addresses which is basically assumed in IP-based discovery systems. On startup, a UPnP device will attempt to obtain an IP address before issuing announcements and queries. The specification recommends the use of DHCP or Auto-IP [33] (when no DHCP server can be reached). With Auto-IP, an entity chooses a random IP address in the range "169.254.x.x/16," verifies that it is not already assigned to an entity in the network (by broadcasting Address Resolution Protocol (ARP) requests) and assigns the address to itself if not already assigned.

Service Advertisement — As shown in Fig. 9, a device announces itself and its embedded devices and services by sending multicast advertisements over the network. These advertisements are associated with a lifetime and contain typically the type of the advertised service or device, and a URL that refers to where the full description of the advertised entity can be retrieved. The number of advertisements that reach a control point is tightly dependent on the time-to-live (TTL) values set for the multicast query and announcement messages (UPnP recommends setting the TTL value to 4, for better performance).

Querying — In addition to service announcements, queries may also be multicasted by control points over the network. They are typically based on the type of the requested device or service. Standard device and service types are assigned by a naming authority (the UPnP Forum by default). When a device receives a query, it matches it against itself and its embedded services and devices. Responses to queries are unicasted to the requesting control point.

Service Handle Retrieval — Similarly to Jini, UPnP provides an event notification mechanism to trigger service updates. UPnP also addresses the requirements of unmanaged and spontaneous networks trough the use of the IP auto configuration mechanism and the use of both queries and spontaneous announcements at the discovery step. However, important features like security and scalability are not considered in the UPnP design. Nevertheless, UPnP has been widely adopted and implemented. It is part of the Microsoft operating system Windows XP, and a few gateway products, including Linksys and D-Link, implement the UPnP technology as well.

WEB SERVICE AND GRID SERVICE DISCOVERY APPROACH

Web Services [1] provide a standard way of interoperating between different software applications, running on a variety of platforms and/or frameworks. Grid Services are specialized *stateful* Web Services with additional Grid interfaces, initially designed for creating a virtual organization where resources and services can be shared seamlessly in a relatively small community distributed across large-scale networks. In this section, we extend our discussion to include Grid Services.

The core of Web Services and Grid Services is the use of XML, SOAP, and respectively Web Service Description Language (WSDL) [34] and Grid Web Service Description Language (GWSDL) in Open Grid Service Architecture (OGSA) [35].

The OGSA adopts a centralized approach for service discovery, whereas service discovery is left open and various service discovery mechanisms are proposed for the Web Service architecture. Universal Description, Discovery and Integration (UDDI) [36], is the de facto standard for Web Service discovery. Many research activities are devoted to enhancing and overriding the legacy UDDI specification for thriving efficiency, scalability and flexibility in the discovery mechanism. Table 2 summarizes some of the proposed architectures for web service discovery. More detailed survey of such activities can be found in [37].

Bootstrapping — The bootstrapping step is not addressed; the location of the directory is assumed to be well-known to the consumers and service providers.

Service Advertisement — Service providers first prepare the XML-based WSDL documents (respectively GWSDL documents in OGSA) which model and describe information about the service before registering it with the directory. In UDDI, the registered WSDL document models information about the organization that provides the service and the service itself (e.g., name, access points, operations). This is done according to the UDDI data model [36]. There are several basic components inside the data model (Fig. 10: BusinessEntity (white

Membership Service	used by the current peer group members to reject or accept a new group membership application. It may enforce a vote of peers or elect a designated group representative to accept or reject new membership requests.
Discovery Service	provides peer, group and other advertisements.
Monitoring Service	allows a peer to get information about another peer's status.
Resolver Service	used to send queries throughout the P2P network. It is used by other services like Discovery Service, to distribute a discovery query to peers involved in the same group.
Endpoint Service	is basically the addressing mechanism used by peers to communicate with each other. Technically, the endpoint service allows a communication to be established between peers using different communication protocols and consequently different address schemes.
Pipe Service	provides a virtual connection between peers involved in a communication.
Rendezvous Service	acts as a proxy for queries over the network. Indexes over PeerGroup advertisements can be distributed among the Rendezvous peers.

■ Table 3. *JXTA core services.*

Peer Discovery Protocol	allows a peer to advertise its own resources and discover the resources provided by the other peers. A resource is represented in a XML document, and published using an advertisement.
Peer Resolver Protocol	implements a request/response protocol that allows a peer to send queries to a specific peer or to the PeerGroup and receive responses to its query.
Peer Information Protocol	allows a peer to obtain information about other peers (e.g., state, traffic load,capabilities).
Peer Binding Protocol	allows a peer to establish a virtual communication channel (i.e., pipe) with one or more peers by binding endpoints.
Endpoint Routing Protocol	used to determine the route between two endpoints, i.e., the intermediary peers that will route a message when there is no direct route between the source and the destination.
Rendezvous Protocol	allows messages to be propagated within a PeerGroup. Within a peer group, peers can be either rendezvous peers or peers that are listening to rendezvous peers. Rendezvous peers are responsible for propagating messages to the associated peers.

■ Table 4. *JXTA protocols.*

pages), BusinessService (yellow pages), BindingTemplate (green pages), and tModel (allows the use of external web service reference such as a WSDL document). UUID keys can be assigned to each business entity, service, binding template, or tModel.

Since Web Services are stateless, registration is done according to a hard-state model in UDDI, whereas in OGSA, registrations are soft-state and each service instance has three time stamps (*goodFrom*, *goodUntil*, *notGoodAfter*) associated with the service data element.

Querying — Queries are keyword-based and follow the XML format. Querying in the UDDI registry approach is very rudimentary and only key lookups with primitive qualifiers are supported. In OGSA, however, more powerful XQuery[1] [44] support has been implemented to facilitate querying and selection.

Responses to queries are ensured to be accurate and complete since the registry has the full control over the information stored inside the registry. The completeness of discovery is guaranteed since all the registry entries can be searched.

Service Handle Retrieval — The result of querying in Web Services and Grid Services is often a list of all service descriptions (in XML format). A set of attributes can be used to describe the business entity and the services they offer. Services then can be invoked through the service handles or access points defined in the UDDI entry. These access points usually take the form of URLs.

In OGSA, the Grid Service Handle (GSH) takes the form of a valid URI [45] that may follow the http-GSH[2] scheme [35]. A GSH is resolved into a WSDL encoded Grid Service Reference (GSR). OGSA uses a factory approach for creating a stateful Grid Service instance. This means each client will be associated to its own service instance. The factory interface for creating a specific service instance is standardized and this information is included in the GSR. A new service instance then can be created and accessed by the customer as described in the GSR.

UDDI and OGSA present the advantage of using standard

[1] *XQuery is a querying language with a similar purpose to SQL, but designed for XML documents rather than relational databases.*

[2] *A GSH may be a URI with an "http" scheme following the http URL syntax. It may and not necessarily be resolved by an http GET mechanism.*

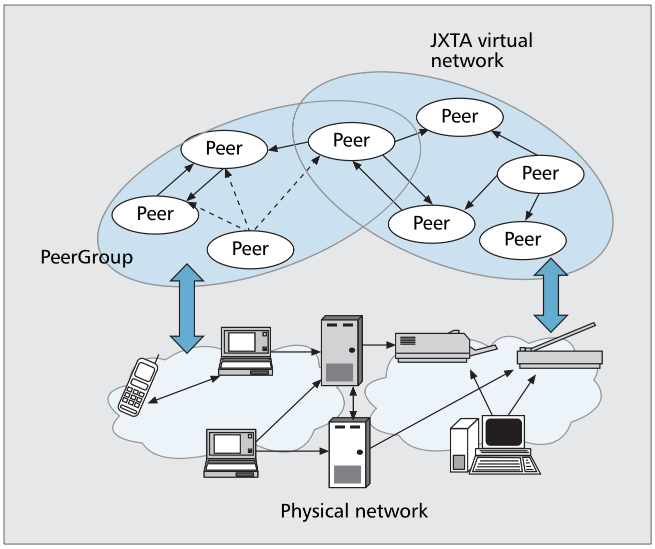


Figure 11. *JXTA architecture.*

and well-known protocols that are independent from the underlying network layer. The increasing interest in Web Services and deployment of Web Service-based solutions in large-scale environments, makes UDDI a promising candidate for incorporating in a multi-domain service infrastructure.

JXTA

JXTA [46] is a P2P platform based on a set of services (Table 3) and six protocols (Table 4) to develop P2P applications. Conceptually, JXTA provides a distributed model for peers to discover each others and interact, and an infrastructure to establish and manage routes and communications between peers across heterogeneous large-scale networks. Peers on top of the JXTA platform operate on an overlay network which provides network transparency (Fig. 11).

Bootstrapping — Bootstrapping in JXTA entails joining a PeerGroup. If the peer is not already configured with a group, it must initiate a discovery process and then subscribe to the group. Typically, on startup, a peer automatically belongs to a generic PeerGroup (WorldPeerGroup or NetPeerGroup) that has a generic implementation of JXTA standard and core services like Peer Discovery and Peer Resolver. These services are used to find the desired group, i.e a first peer belonging to that group. If the peer subscription is accepted by the group, the peer obtains an authenticator (Authentication Credential) that allows it to join the group. At the end of the join step, the peer obtains the membership service as implemented by the group and a credential that the peer will use to authenticate its messages.

Service Advertisement — In JXTA, services are defined by Modules. JXTA associates to each Module three kinds of advertisement: Module Class Advertisement, Module Specification Advertisement and Module Implementation Advertisement. These contain respectively a description of the service behavior, means to invoke the service, or a reference to an implementation of the service (i.e. where to find the code, how to load it, etc.).

JXTA-enabled services are typically published using a Module Specification Advertisement. A Module Specification Advertisement may specify a communication channel through a *pipe advertisement* that must be used by a peer to invoke the service, a list of pre-determined messages that can be sent by a peer to interact with the service, and references to an authenticator and a local proxy for the service.

Querying — If the query cannot be resolved locally, i.e. no match among the cached advertisements, it is then encapsulated in a Peer Resolver query to be sent over the PeerGroup. When a PeerGroup implements the Rendezvous Protocol, queries are sent to Rendezvous Peers. These peers are most likely to cache an index over the requested service advertisement. When a matching index is found, the Peer Rendezvous Protocol is used by the Rendezvous peer to propagate the query to associated peers, using the Pipe Service. This service abstracts the underlying communication model. Even if the underlying transport layer does not support a propagation model (e.g. multicast), the Pipe Service emulates one by initiation several point-to-point communications. However, if the Rendezvous peer can not resolve the query, it propagates it among other known Rendezvous peers (Fig. 12).

Queries are mapped to the cached advertisements in a way that is not specified by the JXTA specification (the method is implementation-dependent). Peers respond to the query with a maximum number of responses as specified in the query.

Service Handle Retrieval — Peers receive in response to their queries a number of service advertisements encoded in XML. The service handle depends on the type of the advertisement.

JXTA provides an open development platform for a service infrastructure on top of large-scale and heterogeneous networks. The peer-to-peer overlay allows the abstraction of the network topology which is very useful to cross domain interactions.

INS/TWINE

INS/Twine [29] was developed by the MIT Laboratory for Computer Science in 2002 as an enhancement of INS.

INS/Twine acts similarly to INS. However, it uses the Chord [43] (Appendix 1) indexing scheme at the INR overlay for better performance and scalability. In contrast to the tree topology used by INS, INRs form a chorded ring that follows the Chord specification.

Bootstrapping — Similarly to INS, clients and service providers use a well-known network address of an INR in the system for bootstrapping.

Service Advertisement — A service provider contacts an INR to register its name-specifier and service information. This INR splits the name-specifier into strands (path from root to leaves), generates a 128-bit numeric key for each strand using the Message Digest (MD5) [47] hash algorithm, and uses Chord to distribute these keys in appropriate INRs along with the information to which the key refers. For a better tolerance for INR failures, a number of replicas of this information are distributed among INRs. The INS/Twine architecture is shown in Fig. 13.

Registrations follow both soft-state and hard-state models. A service maintains soft-state registration with the INR with which it directly communicates. This INR then takes the responsibility of registering/deregistering the service explicitly (i.e. hard-state method) with appropriate INRs in the Chord ring.

Querying — A client constructs a (possibly partial) name specifier (say *n*) based on its request and sends it to a known INR (say Y). Y splits *n* into a set of strands (paths from root to leaves) and converts the longest strand into a numeric key

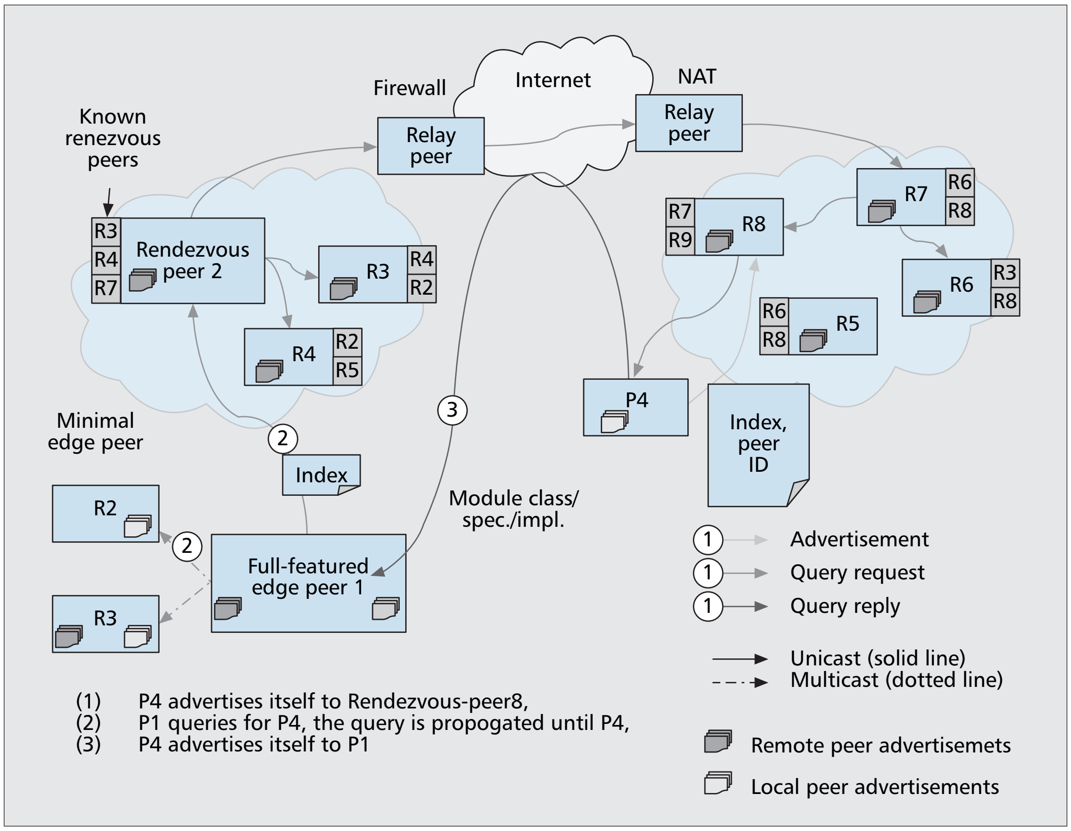


Figure 12. *JXTA discovery process.*

using a hash function. Then it uses a consistent hashing technique to locate an INR (say Z) that possibly holds a resource corresponding to the key (as done in Chord). Now, Z matches n against the name-specifiers in its local database. If matches are found, the discovered service descriptions are returned to the client through Y. Otherwise Y tries to extract another strand form n and iterates the process. Assuming that there are N INRs in the overlay network, and that the level of key replication is K, the query process involves K visited INRs and $O(\log N)$ INRs for key routing [43].

Twine also provides a means for service selection. In this case the INR network returns only one service that matches the query and optimizes an application defined metric.

Service Handle Retrieval — The only access information that Twine provides as a result of the discovery process is the network address for the service. However, a service description, returned as a result of the client's query, can store application-specific data, in addition to the network address.

Twine is an attempt to improve on the design of INS by extending it onto a self-organizing structured peer-to-peer overlay network. By using Chord [43], it claims to enhances the INS performance and scalability by reducing the need for storage capacity, computation, and bandwidth, while reducing the managerial tasks required for management of the overlay network.

SPLENDOR

Splendor [48] has been designed in the Department of Computer Science and Engineering at the Michigan State University, and was presented in 2003. Splendor is a service discovery system that mainly considers security and privacy issues in the service discovery process. Moreover it considers location awareness, and service and client mobility; it mostly targets clients and services connected to wireless mobile networks like 2.5/3G and WLANs.

Splendor is a four-party service discovery mechanism, that involves clients, services providers, directories and proxies. It follows a centralized architecture, where the key element is the proxy. In the service discovery process, a proxy works on behalf of a mobile (i.e., nomadic) service, and performs registration, authentication and authorization for that service.

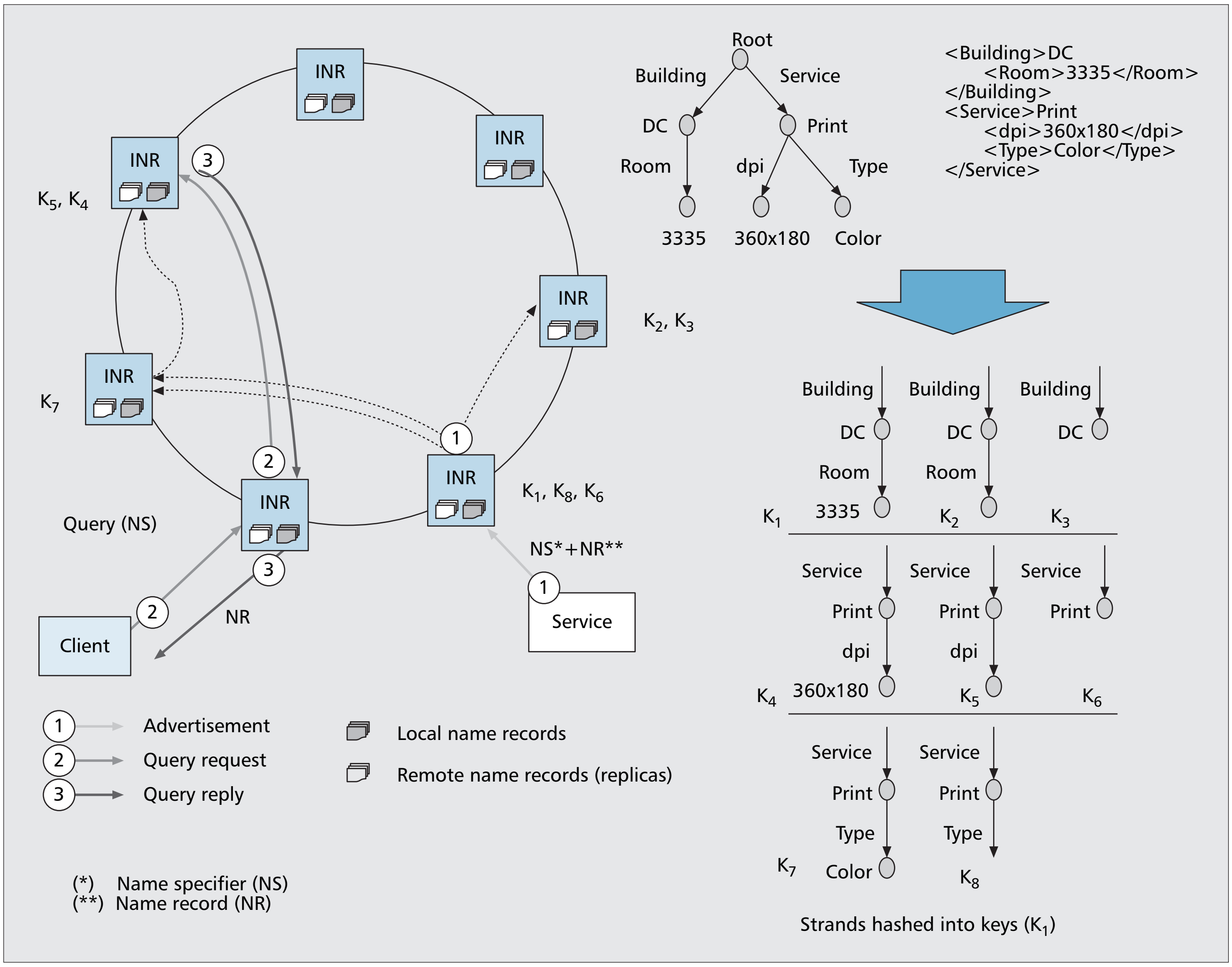
Bootstrapping — A predefined multicast address is used by clients and services for bootstrapping. Directories use this address to periodically multicast their unicast network addresses.

Service Advertisement — As shown in Fig. 14, mobile services do not register directly to the directory. Instead, upon moving into a new network a service contacts its proxy over the Internet and asks to perform registration on its behalf by providing the announced directory address. Splendor uses a tag-based location-awareness, where location-tags are assumed to be emitted by some network components and read by the services in order to track location changes. Messages containing tags may also optionally provide directory addresses and certificates which may be used during the discovery process.

Querying — Clients send their queries to the nearby directory. The querying model and format are left open.

Service Handle Retrieval — Directories respond to client queries with the address of the service proxy.

Splendor conceptually presents an interesting solution especially for public spaces with widespread mobility of both clients and service providers. It does not provide a full system specification in terms of message format or communication mechanism, however it focuses on the security issues. As opposed to the other studied discovery approaches, Splendor defines its own security protocols. It defines the protocols by



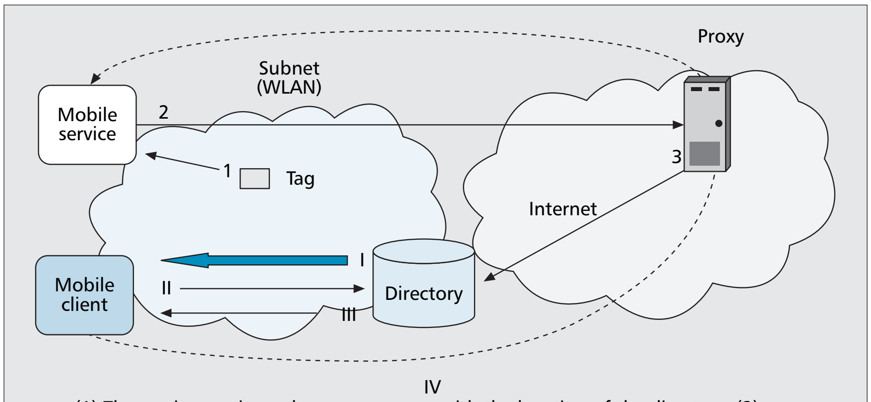
■ **Figure 13.** *INS/Twine architecture.*

which session-keys and certificates are exchanged between entities to ensure mutual authentication, authorization, data integrity and confidentiality. The security framework is further presented next.

Content Sharing Peer-to-Peer Approaches

All content sharing P2P systems offer mechanisms for data content lookup and for data transfer. Although data transfer is usually conducted between two peers, the search mechanism may involve intermediate entities. To facilitate effective search, a file has a name, location, and sometimes description, recorded as an entry in an index file. Search for a file typically involves a query matching on the index file. P2P systems differ in how this index file is distributed over the peers (architecture) and what index scheme is used (index structure). From an architectural point of view (Fig. 15), content sharing P2P systems can be *centralized*, *decentralized*, or *partially decentralized*. This division follows a logical evolution of the P2P systems. **Centralized** P2P systems are characterized by the existence of a central index server, whose sole task is to maintain the index files and facilitate content search. Napster belongs to this category. Centralized P2P systems are highly effective for file search, but the index system itself constitutes a bottleneck and a single point of failure. **Decentralized** architectures remedy this problem by having all peers index their own local content, or additionally cache the content of their direct neighbors. Content search in this case consists in flooding the P2P network with query messages (e.g. through pure broadcast in Gnutella [49]). A decentralized P2P system such as Gnutella is highly robust, but the query overhead is overwhelming in large-scale networks. Recognizing the benefit of index servers, many popular P2P systems today use **partially-decentralized** architectures, where a number of peers (called super-peers) assume the role of index servers. In systems such as KaZaA and Morpheus, each super-peer has a set of associated peers. Each super-peer is in charge of maintaining the index file for its peers. Content search is then conducted at the super-peer level, where super-peers may forward query messages to each other using flooding. The assignment of super-peers is difficult in such a scheme, as it assumes some peers in the network have high capacity and are relatively static (i.e., available most of the time). A newer version of Gnutella also took this approach. We will focus on the original Gnutella design in this section, as it is a representative of the decentralized P2P schemes.

The indexing scheme used by content sharing P2P systems can be categorized as *unstructured*, *semi-structured*, or *structured*. **Unstructured** P2P systems use flat index files, where each index file has no relation to other index files. Napster, Gnutella, and KaZaA/Morpheus belong to this category. **Semi-structured** P2P systems, such as Freenet [50], use a local routing table at each peer. A search is based on filenames that are hashed to binary keys. The query is routed at each peer to



(1) The service receives a beacon message with the lcoation of the directory, (2) sends an advertisement request to the proxy, (3) the proxy registers the service with the directory.

(I) The client receives the directory advertisement, (II) sends its query, (III) receives in response the address of the proxy, (IV) contacts the service via the proxy.

■ **Figure 14.** *Splendor architecture and discovery mechanism.*

the closest matching key found on the local routing table. To prevent infinite querying, a time-to-live value is used. Such mechanism is effective assuming the file content is well replicated over many peers. However, it is virtually impossible to enforce data consistency for file updates. **Structured** P2P systems are specialized in efficient content search using fully distributed routing structure. Examples of this approach include P2P systems that use distributed indexing and querying schemes, such as Chord [43], CAN (Content Addressable Network) [51] and Tapestry [52] (Appendix 1). The INS/Twine overlay network formed by INRs is a typical example of structured P2P system. Since this approach has been previously described, in this Section we will only consider unstructured and partially-structured approaches.

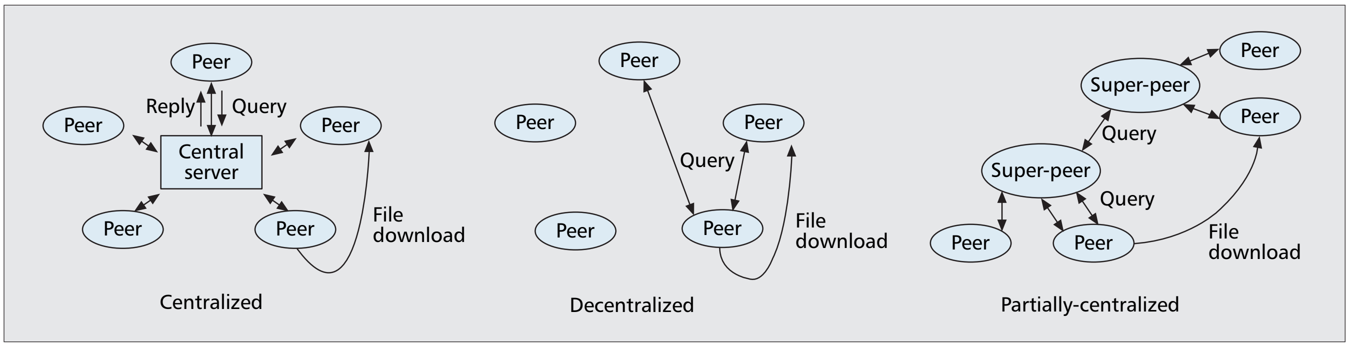
Bootstrapping — When bootstrapping, Napster and KaZaA assume that the index server is on well-known location (this is a configuration parameter in the software). A single connect message is sent by a peer to a index server on join. The peer is fully connected to the P2P network afterwards. Freenet and Gnutella use flooding based neighborhood discovery. Freenet floods the network in search of a single peer node that is connected to the network, while Gnutella floods the network in search of all of its direct neighbors. After this process, the peer is fully connected to the P2P network.

Service Advertisement — With the exception of Gnutella, a data source (service provider) is responsible for registering its data content with the directory. In Gnutella, as the directory information is stored locally (where the data source is stored), there is no registration process. The directory information is stored centrally in Napster, distributed among super-peers in KaZaA, kept locally in Gnutella, and distributed non-deterministically in Freenet. The index distribution is non-deterministic in Freenet as the data is inserted n hops away from the insertion request, based on routing hints given along the path that leads to nodes with similar hash keys (there may exist many nodes with similar hash keys and the data itself is replicated multiple times in the network).

Querying — Napster sends query messages directly to the index server, whereas Gnutella floods the P2P network with query messages since each peer sends queries to all of its immediate neighbors. The scope of flooding is controlled by the time-to-live value of the query message. The query will be propagated onwards until a peer holding the requested data is found or the TTL expires. A KaZaA peer sends the query message directly to its super-peer, which then proceeds to flood other super-peers with the query message. Freenet, uses peer-routing for query lookup. A single query message is forwarded from each peer based on its local routing table information. When using indirect files in Freenet, more than one message may be sent out from a peer (looking for multiple files). All of these P2P systems perform text-based matching on either filenames or keywords. Selection is handled very primitively in Napster, Gnutella, Freenet, and KaZaA by restricting the number of hits returned. The hits themselves may be ranked by number of access, number of keyword matches, etc.

Service Handle Retrieval — In response to the query, with the exception of Freenet, requesting peers receive the contact information (i.e., IP address) of the peer that holds the requested data. Freenet will send the requested data as part of the query result. This design is chosen for data source anonymity (i.e., the data source is not known to requester) and to utilize peer caches (i.e., any peer along the querying path that contains the requested data may resolve the query locally).

With the exception of Gnutella whose flooding mechanism does not fit large-scale networks, the studied approaches generate very low overhead and are designed for the Internet scale.



■ **Figure 15.** *Content sharing P2P architectures.*

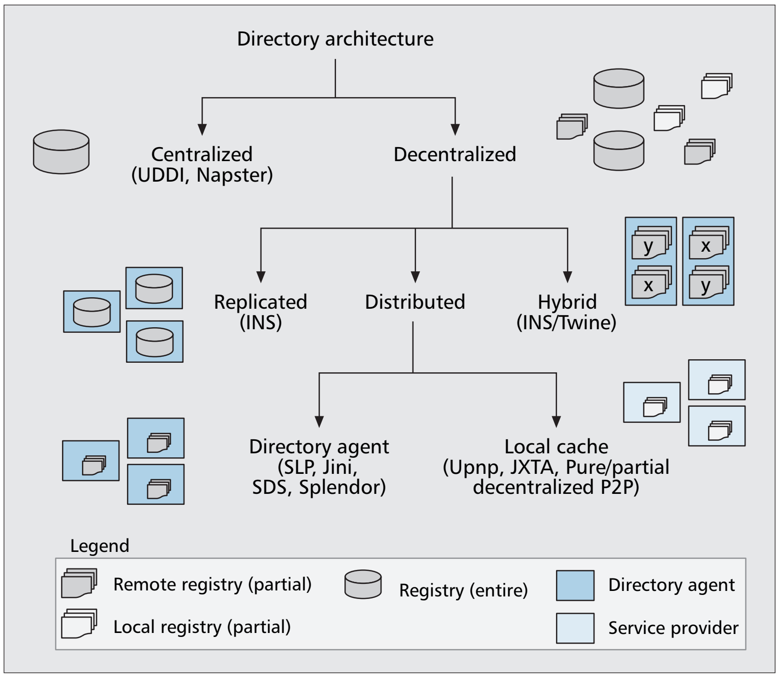


Figure 16. *Service discovery system architectures.*

Technology Comparison

In this section, discovery approaches are evaluated and compared with respect to the chosen criteria. A summary of the comparison has been placed in Table 5, Table 6, and Table 7.

Architecture

The architecture adopted by different service discovery approaches can broadly be classified as centralized and decentralized (Fig. 16) according to how the directory information is stored. In the former category, a dedicated directory or registry maintains the whole directory information (as in Napster and centralized UDDI), it takes care of registering services and answering to queries. In the second category, the directory information is stored at multiple network locations. Decentralized systems can be categorized as *replicated*, *distributed*, or *hybrid*. In the replicated case, the entire information is stored at different dedicated directories (as in INS). In the distributed case, the directory information is partitioned. Directory information is stored at different network locations; it is either stored in dedicated servers, i.e. directory agents (DA) (as in SLP, Jini and SDS where a hierarchical architecture is used) as per a three-party model or cached locally by the service providers in the system (e.g., UPnP, JXTA, and centralized P2P systems) according to a two-party model. Finally, in the hybrid case, both replication and distribution are used (as in INS/Twine).

In large-scale networks, a centralized directory becomes a performance bottleneck and a single point of failure that may lead to the crash of the whole system if no recovery mechanism is deployed. Consistency of the replicas is a major issue in the replicated architecture (like INS), since maintaining consistent replicas is usually bandwidth-consuming. On the other hand, when the directory information is distributed, i.e. partitioned, among dedicated directories, the failure of one of them leads to the unavailability of a part of the directory information. The fully distributed two-party architecture attempts to remedy all these issues, however these systems generally does not scale well, since they use multicast-like communications which are expensive in terms of bandwidth.

Decentralized hybrid architectures seem to offer the best compromise between bandwidth consumption, hence scalability, and fault-tolerance. INS/Twine falls into this category.

INS/Twine presents the optimal architecture that conciliates fault-tolerance (using INR replicas for tolerance to INR failure) and scalability (directory information is distributed in a Chord ring).

Effectiveness

In SLP, Jini, UPnP and Salutation, the correctness of the response to a query is ensured by the use of an exact pattern matching during service filtering and selection, assuming the directory information is valid. Splendor, Web Services and Grid Services use a similar approach for ensuring correctness.

SLP and UPnP uses administratively-scoped multicasting [53], which inhibits multicast packets from crossing the configured administrative boundary. This ensures completeness of the discovery process within an enterprise network, as targeted by these approaches. However, for Jini time-to-live scoping may impede completeness; for complete discovery, it must have a TTL large enough to cover the whole discovery network. In Salutation, the completeness of the response is ensured via the capability exchange mechanism used between Salutation Managers.

SDS uses Bloom filters for representing service descriptions. Possibility of false positives is inherent in Bloom filters, making it possible for a client discover non-matching service descriptions as a result of a query, if the size and load of the Bloom filters are not chosen appropriately. However, it is ensured in SDS that all the advertisements matching a query can be discovered, despite the possibility of having some redundant results.

In Splendor, Web Services and Grid Services, the completeness of the query response is assured by, however restricted to, the centralized registry; query responses will not contain those service descriptions stored within registries that are not reached by the query.

In case of INS and INS/Twine, directory information replicated in different INRs remains consistent with infrequent updates (join and leave of INRs; state change of services). Responses to queries are expected to be complete in such environment. On the other hand, correctness of the discovery process depends on the placement of attributes in the query (name-specifier tree). A client requires prior knowledge of the advertised attributes and their levels in the name-specifier tree.

For content-sharing P2P systems, Gnutella and Freenet cannot guarantee correctness of the query response. Effectiveness in JXTA mainly depends on the querying method used.

Solutions related to exact pattern matching (such as in SLP, Jini, etc.) offer guarantees on the correctness of the query response. Systems with dedicated and conceptually centralized repositories, such as UDDI, guarantee query completeness within the scope of the queried repository. Structured P2P systems like DHTs, guarantee query completeness within the scope of the distributed directory system. A system like INS/Twine with an enhanced querying model (such as pattern matching) for correctness guarantees would meet the effectiveness requirements.

Criteria		Salutation	SDS	SLP	Jini
Architecture		Decentralized two/three-party	Decentralized three-party	Decentralized two/three-party	Decentralized two/three-party
Effectiveness	Correctness	Guaranteed (exact pattern-matching)	Possible false-positive	Guaranteed (exact pattern-matching)	Guaranteed (exact pattern-matching)
	Completeness	Guaranteed	Guaranteed	Guaranteed in administrative scope	Limited by TTL
Fault Tolerance	Point of failure	Dedicated SM in 3-party setting	SDS server near root level	DA in 3-party setting	LUS in 3-party setting
	Recovery Methods	Periodic availability check for srv	Periodic multicast announcements	Turn to 2-party, soft-state srv advertisement	Turn to 2-party soft-state srv advertisement
Performance	Communication Overhead	Moderate	Moderate	Moderate	Moderate
	Load Balancing	Not considered	Child servers spawned	Not considered	Not considered
Security	Integrity	Not considered	Not considered	Digital signatures	Digital signatures
	Privacy	Not considered	Hybrid encryption	Recommended IP/ESP	Encryption
	Authentication	Primitive login/pwd	Certificates	Assumed within the domain	Certificates, TLS
	Authorization	Not considered	Client capability list	Not considered	Client permission policies
Platform Network Independence	Platform dependence	Independent	Independent	Independent	Java platform
	Network dependence	Independent	IP	IP	IP
Scalability		LAN scale	WAN scale	LAN scale	LAN scale
Interoperability with Other Discovery Services		Possible, e.g., SLP	Not considered	Possible, e.g., Jini, Salutation	Possible, e.g., UPnP, INS/Twine
Standardization and Existing Implementations		de facto standard, implementation exists	Research work	Standard, implementation exists	de facto standard, implementation exists

■ Table 5. *Comparison of discovery systems (1).*

Fault Tolerance

Centralized architectures (like UDDI registry and Splendor) suffer from a potential single point of failure. Using multiple directory agents in SLP and multiple lookup services in Jini improves the resilience of the system by removing the single point of failure. Jini employs peer-lookup for discovery in case of lookup service failure, while SLP user agents reattempt to discover a new Directory Agent, and if no Directory Agent can be reached, multicast requests are sent to Service Agents.

Content-sharing P2P systems vary significantly in the ability to handle faults. Napster has a single point of failure on its index server, but no recovery method is used. The failure of super-peers in KaZaA results in a network partition. Gnutella does not use any dedicated registry, resulting in better robustness.

In INS, the failure of an INR node can lead to tree partitioning. The replication of the whole directory information in all INRs allows the discovery system to recover this failure, however this leads to the failure of update propagation. In INS/Twine, the recovery from INR failure is handled by Chord.

To handle failure of SDS servers, SDS adopts multicasting. SDS servers listen on a well-known multicast channel for periodic service advertisements and announcements from other SDS servers. The absence of announcement from a SDS server indicates its failure. In that case, another SDS server in proximity takes over the place (domain) of the failed server and announces itself as responsible for that domain.

In addition to node failures, communication failures may also impede the functioning of the service discovery system since they introduce inconsistency in the system. Some studies [54, 55] have tested and measured the performance of the self-healing techniques of discovery approaches (for instance Jini and UPnP) in response to message loss and interface failures respectively. The Jini and UPnP specifications distinguish

Criteria		INS	UPnP	Splendor	Grid Services
Architecture		Decentralized replicated	Decentralized	Decentralized three-party	Centralized
Effectiveness	Correctness	Depends on the position of av-pair	Guaranteed (exact pattern-matching)	Implementation-dependent	Guaranteed
	Completeness	Guaranteed	Guaranteed in administrative scope	Implementation-dependent	Guaranteed
Fault Tolerance	Point of failure	INR	SAs	Directory	Directory
	Recovery	Replication, Soft-state advertisement	Soft-state srv advertisement	Soft-state srv advertisement	Soft-state srv advertisement
Performance	Communication Overhead	High	High	Low	Low
	Load Balancing	Child node spawned	Inherent to the architecture	Not considered	Among service instances
Security	Integrity	Not considered	Not considered	Digital signatures	Digital signatures
	Privacy	Not considered	Not considered	Encryption	Encryption
	Authentication	Not considered	Not considered	Certificates	Certificates
	Authorization	Not considered	Not considered	Credentials	Credentials
Platform Network Independence	Platform dependence	Independent	Independent	Independent	Independent
	Network dependence	IP	IP	IP	Independent
Scalability		WAN scale	LAN scale	WAN scale	Internet scale
Interoperability with Other Discovery Services		Not considered	Possible, e.g., Jini	Not considered	Possible
Standardization and Existing Implementations		Research work, implementation exists	de facto standard implementation exists	Research work, implementation exists	de facto standard, implementation exists

Table 6. *Comparison of discovery systems (2).*

two techniques for consistency maintenance: *notification* and *polling*. In polling, a user agent periodically sends queries to obtain information about a service that was previously discovered, in order to update the service description that was previously retrieved and cached locally. In notification, the user agent subscribes to receive events announcing that a change has a occurred in the description of a specific service. The studies show that a two-party architecture with the polling technique (like UPnP) is the most effective, i.e. offers the highest probability that a user agent will receive a change when it occurs at the desired service, however a three-party architecture (with a single repository) with notification (like Jini with a single LUS) is slightly more efficient than a two-party architecture with the polling technique, i.e. propagates fewer messages on the system to indicate changes in the service.

When a failure is detected while attempting to contact a node (either due to communication failure or node failure), two techniques may be used (separately or in combination) by discovery systems to recover [56]. In the first technique, called *application-persistence*, a number of attempts may be issued to re-contact the node. On failure, the information about that node will be discarded from the cache. In the second technique, the information about the node is stored in the cache with a life-time as per a *soft-state* discovery model (like in Salutation, SDS, SLP, Jini, UPnP, INS/Twine); the information is discarded from the cache once the life-time expires without receiving a re-announcement from the node. When these techniques are combines, attempts to reach a node will stop as soon as the number of maximum attempts is reached or the life-time of the node information expires. This solution generally leads to optimal results. It is used by UPnP, SLP and Jini for contacting the SA, DA and LUS respectively.

UDDI addresses call failures by establishing a calling convention (*retry on failure*) that relies on the use of cached service *bindingTemplate* information, refreshed from the UDDI registry whenever a call failure occurs. The bindingTemplate, first obtained from the UDDI registry, is cached and then

Criteria		INS/Twine	JXTA	Web Services (UDDI)	Partially-centralized and Structured P2P
Architecture		Decentralized hybrid	Decentralized two-party	Varies, see Table 2	Decentralized three-party
Effectiveness	Correctness	Depends on the position of av-pair	Guaranteed	Implementation-dependent	Guaranteed
	Completeness	Guaranteed	Implementation-dependent	Restricted to queried directory	Guaranteed
Fault Tolerance	Point of failure	INR	Peers	Directory	Index nodes
	Recovery Method	Chord	Periodic updates	Retry on failure, cache update	Periodic updates
Performance	Communication Overhead	Moderate	Moderate	Low	Moderate
	Load Balancing	Inherent to Chord	Not considered	Not considered	Among super-nodes
Security	Integrity	Not considered	Digital signatures	Digital signatures	Not considered
	Privacy	Not considered	Encryption	Encryption	Not considered
	Authentication	Not considered	Certificates, TLS	Certificates, Kerberos tickets	Not considered
	Authorization	Not considered	Credentials	In progress	Not considered
Platform Network Independence	Platform dependence	Independent	Independent	Independent	Independent
	Network dependence	IP	Independent	Independent	IP
Scalability		WAN scale	Internet scale	Internet scale	Internet scale
Interoperability with Other Discovery Services		Possible, e.g., Jini	Possible	Possible; protocol stack	Possible
Standardization and Existing Implementations		Research work, implementation exists	Standard, implementation exists	de facto standard implementation exists	Research works, implementation exists

Table 7. *Comparison of discovery systems (3).*

used to call the associate remote Web Service. When a failure is detected, a newer version of the bindingTemplate is obtained from the UDDI registry, the service is called again, and if the call succeeds, the cached bindingTemplate is replaced with the new one.

In terms of robustness, distributed two-party architectures with polling technique (like UPnP) seem to offer a good solution for the tolerance to node failure and the maintenance of system consistency. Jini, SLP, INS and INS/Twine can also handle node failure reasonably well thanks to their recovery techniques based on the soft-state discovery model.

Performance

It is very difficult to evaluate the performance of the studied systems especially that there are no comprehensive benchmarks or evaluation measurements of service discovery approaches; existing works (like [9]) do not cover all the systems, and it is especially difficult to attempt to evaluate them in a large-area setting. We will therefore focus on two broad criteria: communication overhead and load balancing, and try to qualitatively compare the performance of different approaches.

Communication Overhead — The communication overhead often depends on design choices and considerations. Following are the major conceptual choices that influence the network load.

- **Soft-state vs. hard-state advertisements**: Approaches where a service advertises itself as per a soft-state model, like SDS, SLP, Jini, UPnP, INS, INS/Twine, OGSA and JXTA, require announcement or registration renewal before the lifetime of the service expires. This certainly generates more communication overhead than the hard-state model. However, first, this overhead is not significant if services have reasonably long lifetimes (in range of hours and days), and second, soft-state advertisements prevent managing and maintaining the consistency of caches when a service fails before proceeding to an explicit deregistration.

- **Multicast vs. unicast**: Multicast and flooding systematically generate communication overhead; entities are likely to receive multicast/broadcast messages even when they are not interested in. UPnP uses multicast for both queries and announcements, and Gnutella uses flooding; both are bandwidth-consuming. In SLP, Jini and Salutation, bootstrapping may imply multicasted or broadcasted messages, respectively, for user and service agents to find a directory agent, for clients and service providers to discover a lookup service and finally for Salutation managers to discover each other. In SLP and Jini, the overhead increases when no directory can be reached. Nevertheless, it is worth noting that SLP, Jini, and UPnP specifications recommend the use of low time-to-live (TTL) values for multicasted messages, e.g. UPnP recommends 4 as opposed to 15 -default TTL value for Jini- or 255 -default value for SLP-. This basically restrict the multicast scope and hence lightens the network load. However, that may partition the network: when a query is multicasted with a low TTL, it may not reach all service providers and hence the query result may be incomplete. SDS also uses multicasting for service advertisements and announcements from SDS servers (usually in LAN environment). However, client-SDS server and inter-SDS server communication is over unicast channels. The overall communication overhead in SDS is controlled by restricting the multicast communication in local domains of SDS servers (usually LAN) and utilizing unicast communication for wide area connectivity between SDS servers.
- **Centralized vs. decentralized architecture**: In general, a service discovery approach using a centralized architecture has less messaging overhead than a service discovery approach using a decentralized architecture. A caching mechanism may be used by decentralized systems, like UPnP and Salutation, to lighten the network load. However, when services are advertised with a hard state, like in Salutation, the cache is not ensured to be consistent. In INS and SDS, the use of replication also prevents queries from being automatically spread into the whole network. However, the replication mechanism implies the propagation of new information and advertisement updates through the whole or part of the network, which is bandwidth-consuming as well. As opposed to INS where directory information is replicated in all INRs, the number of replicas for an INR in INS/Twine is a fixed system parameter which restricts the communication overhead due to the maintenance of replicas. On the other hand, due to the use of Chord, the maintenance of the ring topology of INRs in INS/Twine involves additional message exchanges however allows to upperbound the number of participating nodes in query and advertisement routing. The use super-peers in KaZaA and Rendezvous peers in JXTA is a good compromise in terms of bandwidth consumption.

In terms of communication overhead, UDDI appears an optimized approach since, first it uses a centralized architecture, it uses only unicast communications (even the directory is not assumed to be discovered at bootstrapping; its location is assumed to be well-known), and finally services are registered with hard state. For system consistency maintenance, as discussed previously, it is preferable for a discovery system to implement the soft-state model. In addition, for the sake of scalability and robustness, the centralized architecture is not advised. INS/Twine presents an interesting compromise as besides its architecture and robustness, it allows to upperbound the communication overhead.

Load Balancing — Centralized approaches often suffer from overload problems as the number of services registered with the registry and the number of consumer queries increase. Load balancing in that case can be handled through replication; i.e. through introducing a multi-node registry. UDDI, SLP and Jini allow the deployment of this kind of mechanism. In SLP and Jini more than one directory agent, or lookup service, can be deployed within a scope, or djinn, respectively. However, none of the three systems have yet considered how to balance registry load.

In INS, when a resolver is overloaded, it may automatically spawn an instance on an other candidate resolver (currently inactive), and then terminate itself if it is not loaded anymore. Information about the candidate-node is retrieved from the DSR, and once the INR load falls below a threshold, it informs its peers and the DSR before terminating itself.

To reduce/distribute load at the root node of SDS server tree, queries are propagated horizontally among siblings, when it reaches near the root SDS server.

In general, discovery systems are either inherently loadbalanced (like Freenet) or do not well address this issue. INS is in reality the only system that addresses load-balancing. Thanks to their inherent load-balancing feature due to the use of consistent hash functions for data indexing, DHTs are very attractive for the design of a distributed directory system.

SCALABILITY

Salutation, SLP, Jini and UPnP are designed for relatively small networks, typically local area networks, where a directory may not be required. For larger deployment, the use of directories and low TTL values for multicast communication is recommended by SLP and Jini. However, even with low TTL, UPnP still does not fit large networks because of the overhead generated by the multicast communications. On the other hand, Salutation's scalability is tightly dependent on the way Salutation managers know about each others and exchange capabilities. The classical way these tasks are achieved, i.e. through broadcast or RPC broadcast [17], cannot be performed in large networks. For a better scalability, the Salutation Consortium is pushing towards the expansion of the Salutation architecture to support directory-based service discovery. A framework for interworking Salutation with SLP is currently being finalized.

INS presents the disadvantage of being computation-intensive and bandwidth-consuming when the number of handled services is large. This is respectively due to the resolution and the propagation processes. With the use of the Chord indexing scheme, INS/Twine presents better performance in large-scale networks. INS/Twine is assumed to scale well into networks involving up to $O(10^8)$ services and $O(10^5)$ resolvers [29].

Although UDDI and Splendor are designed for wide-area networks, their centralized architecture is not suitable for the Internet-scale. Similarly, although SDS aims to support a large user base, the architecture relies on a few central components (i.e. Certificate Authority and Capability manager) which reduces its suitability as a secure wide-area discovery system.

Peer-to-peer approaches, like JXTA and content sharing systems, are designed to fit large-scale Internet-like networks, with the exception of Gnutella because of its flooding mechanism. Generally, peer-to-peer approaches offer good solutions for scalability and self-organization.

SECURITY

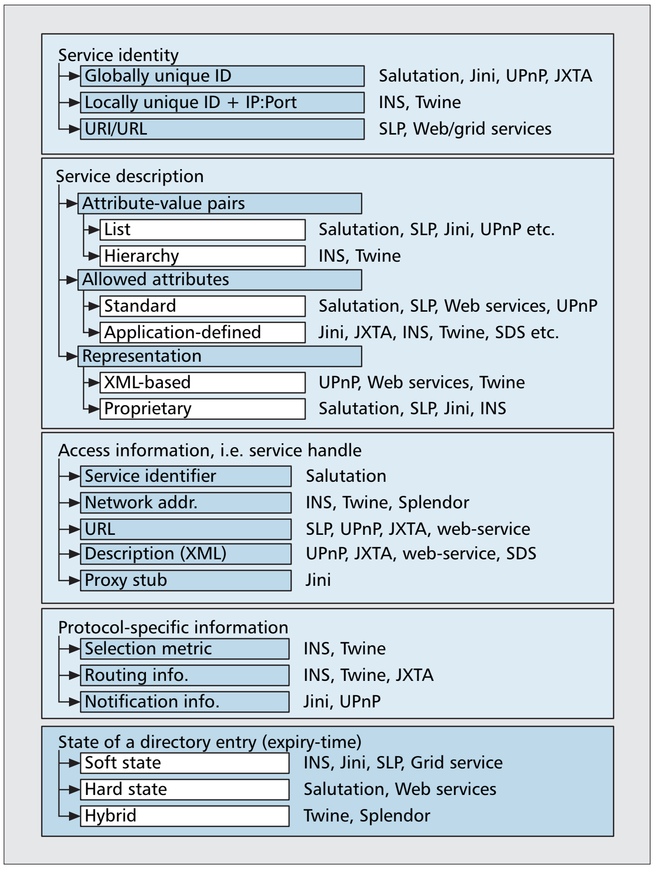


Figure 17. *Directory information: content and data formats.*

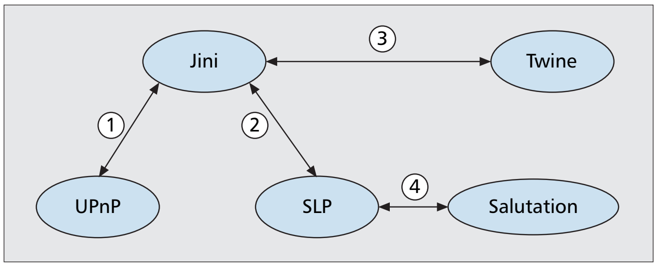


Figure 18. *Existing approaches for interworking service discovery technologies: (1) Jini-UPnP, (2) Jini-SLP, (3) Jini-Twine, (4) Salutation-SLP.*

The security issue is not considered in most of the service discovery systems, for example in Salutation (where simple login/password scheme is provided between clients and functional units which is basically not part of the discovery process), UPnP, INS, INS/Twine and most of the content sharing peer-to-peer approaches. It is weakly addressed in KaZaA where a simple login/password authenticate the users, and in Freenet which conceptually provides user's anonymity. Security is considered in more depth in SDS, SLP, Jini, Web Service, OGSA, JXTA and Splendor. Authentication, authorization, data integrity and privacy are the major security issues that are completely or partially considered by these systems.

SLP considers message integrity by providing Authentication Blocks (i.e. digital signatures) over service URLs and service attributes in the advertisement messages. This allows user and directory agents to check the integrity of advertised service URLs and attributes. Entities that generate authentication blocks are those which have been configured in advance by administrators. User agents verify signed data coming from directories and service agents and assume that they are "trustworthy." This "trustworthiness" relies on the administrators to ensure the secrecy of cryptographic keying for service and directory agents. SLP does not provide confidentiality but recommends the implementation of IP Encapsulating Security Payload (ESP) [57] when it is needed to provide confidentiality for exchanged messages. SDS considers authentication, authorization and message confidentiality for multicast communication. It offers authentication for client-SDS server, service-SDS server, and inter- SDS server communications. It can control the visibility of service descriptions to only the authorized set of users. For multicast communications, message confidentiality is ensured by adopting a hybrid encryption technique, for performance reason. Each multicast message is encrypted using a symmetric key and the key is embedded in the message header. The message header is then encrypted using the public key of the receiving entity.

Jini, OGSA (and more exactly the Grid Security Infrastructure (GSI)), JXTA and Splendor use similar approaches to cover the security issue. They use mutual authentication between the different involved entities in the discovery system; GSI and Splendor use X.509-based certificates [58], Jini allows the use of Transport Layer Security (TLS) [59] and other mechanisms, and JXTA protocols are designed to be compatible with TLS, in addition, certificates can easily be added to JXTA messages. Moreover Jini, GSI, JXTA and Splendor use credentials to ensure authorization, public-key cryptography (a Public Key Infrastructure (PKI) [58] for instance) for confidentiality and digital signatures to ensure data integrity.

The Organization for the Advancement of Structured Information Standards (OASIS) consortium has elaborated and promoted the "Web Services Security" (WSS) as a standard Web Service security model [60–62]. The WSS framework relies on existing security technologies like the XML Digital Signature, XML Encryption and X.509 certificates for ensuring message confidentiality and integrity. It defines a set of SOAP extensions (for data integrity and confidentiality), describes the way Kerberos-like tickets can be used for user authentication and the way X.509 certificates can be used in combination with SOAP extensions for message authentication.

As in distributed systems, establishing end-to-end security guarantees in service discovery systems is very challenging in large-scale settings; the number of malicious users and service providers is more likely to increase with the size of the network, and it becomes harder to ensure that a message that traverses multiple domains is "safe" in all parts of its route. In large-scale and multi-domain settings, a discovery system should protect itself from malicious users and service providers by establishing an authentication mechanism between all involved entities. In addition, in order to ensure message security, it is important to prevent data from being altered and sensitive information from being captured by malicious third parties. In fact, 1- a malicious intruder can capture a response to a query, replace the access information of a "safe" service provider by the access point of malicious one, and without message integrity check, the user will not be able to detect the attack, 2- a malicious intruder can use a captured access information to attack the associate service provider (Denial of Service attack). Note that authorization is an extra security level, mostly required to rule the way users access "personalized" service depending on the granted access rights. In a small-network and a single administrative domain an SLP-like security mechanism can be sufficient. However, this mechanism that assumes the trustworthiness of users and service providers within the domain boundaries, becomes obsolete in large-scale and multi-domain setting. For the sake of interoperability, it is preferable to use standard security mechanisms and well known techniques like X.509 certificates for mutual authentication between involved entities, public-key cryptography for message confidentiality and digital signatures for data integrity. In addition, in large-scale and multi-domain settings, it is important to rely on public security authorities (like VeriSign), which are not used by Splendor or SDS. Jini, JXTA, GSI and Web Services provide a good level of security and use standard security mechanisms and architectures compliant with a large-scale setting. Within these frameworks, implementing WSS and SOAP extensions would be required for securing the envisioned inter-domain web-based discovery mechanism.

Platform and Network Independence

With the exception of Jini which requires a Java-based implementation and, hence, commonly relies on a Java Virtual Machine (JVM), all of the examined approaches are programming-language independent.

SLP, UPnP, INS, INS/Twine, Splendor and most of the content sharing peer-to-peer systems are designed for IP networks. UPnP also assume the support of IP multicast. Since the protocol details of SDS are not publicly available, its only nod towards system-independence is its use of XML for describing service advertisements. Web and Grid services use standard protocol stack, which makes them independent of operating system platform and development language. Since UDDI relies on HTTP, it has the added advantage of not requiring reconfiguration from most firewalls. Together with Salutation and JXTA, Web and Grid services explicitly tackle platform-independence, and hence offer the best solution in terms of network and transport layer abstraction.

Interoperability with Other Discovery Services

Most of the service discovery approaches mentioned in this survey are not interoperable with each other. The use of specific communication protocols, data and data formats (Fig. 17) prevent such interoperation.

Jini, for example, is difficult to interoperate with other approaches due to Jini's exclusive use of Sun technologies (e.g RMI and Java Classes) for communication and description. Similarly, the naming scheme adopted in INS (and INS/Twine) is not standard, and it is difficult to incorporate such a naming scheme in other service discovery approaches, like SLP or Jini. In Splendor, the use of specific protocols for authentication and message encryption prevents from interworking with other service discovery approaches. The absence of details on the SDS protocol makes it hard to evaluate its interoperability with other discovery systems.

None of the analyzed P2P systems interoperate with each other. This is mainly due to their lack of standard directory information representation and communication protocol. If these architectures are implemented using a standard development suite such as JXTA, and provided with a standard definition of directory information, then interworking among P2P systems is possible. However, the unique indexing and routing schemes used by some P2P systems makes their interworking with other discovery approaches difficult.

On the other hand, UPnP is designed to be open by using well-known and standardized technologies. It is then feasible for other discovery mechanisms to interwork with UPnP. Similarly, since SLP is standardized, simple, and uses well-known features like LDAP filters, DHCP and IANA assigned names, other approaches could be enhanced to interwork with SLP. For example, Salutation Consortium is working on a framework to make Salutation interoperate with the SLP system to resolve scalability issues [63].

Web Services are meant for interoperability in the Internet. With the help of XML, SOAP, and WSDL, Web Services are expected to be interoperable with other service discovery mechanisms. Of course, well-defined and well-accepted XML schemas have to be defined. OGSA does not address the inter- Grid interaction in detail. Also, although Grid Services are generally considered as stateful Web Services, Grid Services cannot interact very well with Web Services. Recently, there has been some work to merge Web Services with Grid Services. Example proposals include the Web Service Resource Framework (WSRF) [64], which focuses on building stateful Web Services.

As shown in Fig. 18, a number of works bridging Jini- UPnP [65], Jini-SLP [66], Jini-Twine [67] and Salutation- SLP [63]) have been conducted to enable interoperation between two different service discovery technologies. Each of these work employs a kind of "bridge" for protocol and data conversion.

Koponen *et al.* [66] 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 the 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 a performance bottleneck.

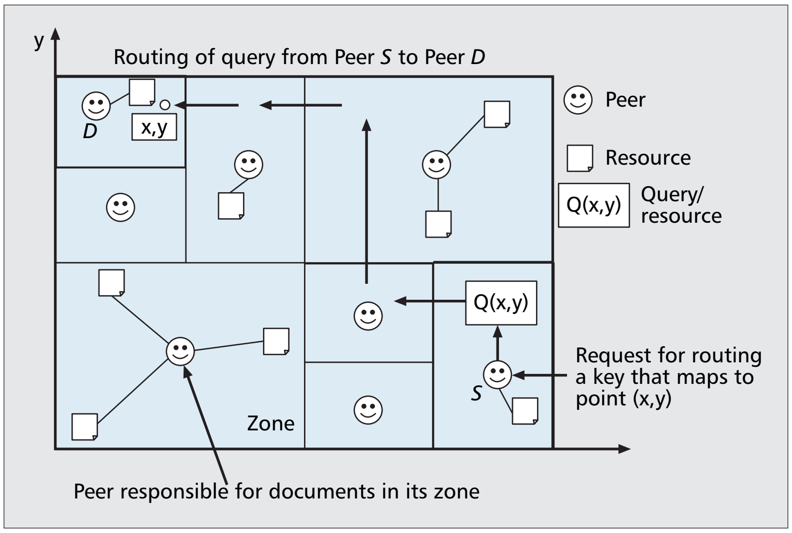
Another work [65] presents an architecture for interworking Jini and UPnP, where virtual clients and services are placed in each domain. For a service that is discovered by a virtual client in one domain, a corresponding virtual service is created in the other domain. The virtual service registers itself 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 since all the services of all domains are mirrored in each domain.

A different approach [67] 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.

Despite attempts at interoperating several of the service discovery systems, among all the approaches, Web Services may be the easiest one to extend in order to address the interoperability issue, as it uses standardized Web technologies (e.g. XML, SOAP, and WSDL).

Standardization and Existing Implementation

Several international organizations have been involved in standardizing or promoting service discovery approaches.


y
Routing of query from Peer S to Peer D
D
x,y
Peer
Resource
Q(x,y)
Query/resource
Q(x,y)
Request for routing a key that maps to point (x,y)
S
Zone
Peer responsible for documents in its zone

■ **Figure 19.** *Example of CAN lookup in 2 dimensions.*

Specifically, they are the IETF for SLP, Globus Alliance for OGSA Grid Services, the W3C and UDDI Consortium — a section of the Organization for the Advancement of Structured Information Standards (OASIS) — for Web Services/XML/SOAP/WSDL/UDDI, Salutation Consortium for Salutation, and UPnP Forum for UPnP. On the contrary, SDS, INS and INS/Twine remained in the stage of research works.

There are a number of existing implementations and development platforms for Salutation, SLP, Jini, INS and INS/Twine, UPnP and JXTA.

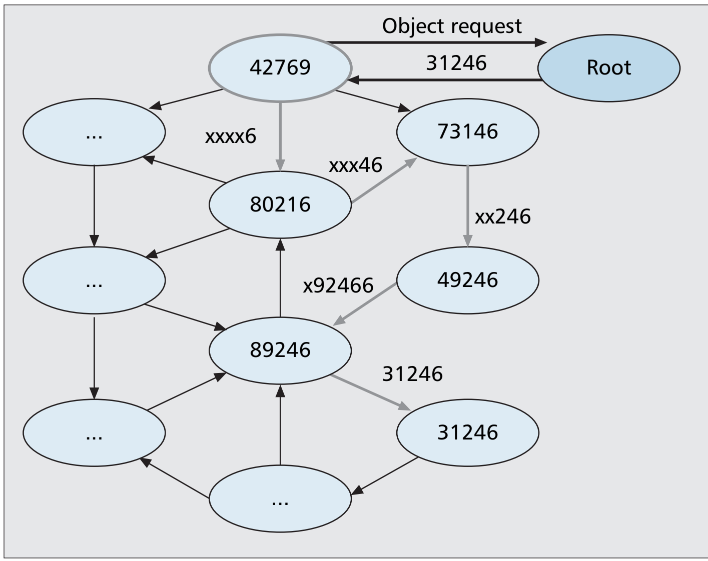
- IBM provides a Salutation Software Development Kit (SDK) and Salutation-Lite is provided in Open Source at the Salutation Consortium web site [18].
- The OpenSLP project [68] delivers an open-source implementation of SLP.
- Sun Microsystems has released the Jini Technology Starter Kit [69] to build Jini-enabled clients, services, and lookup services.
- The current implementation of INS and INS/Twine uses Java 2.0 platform and is available at [70].
- A number of UPnP SDKs are available, including the ones developed by Intel [71] and the open source SDK for the Linux platform [72].
- An implementation of the JXTA Protocols using Java 2 Standard Edition (J2SE) [73], and another one in C for Windows and Linux platforms [74] are available.

WebSphere [75], .NET [76], and SUN ONE web services [77] are the main existing implementations of Web Services. Only Globus provides a development toolkit (Globus Toolkit [78]) to implement Grid Services. The most important standardizations efforts and active standardization bodies target Web-based technologies (like Web services). For the sake of interoperability and standardization, web technologies are definitively the best choice.

TECHNOLOGY SELECTION

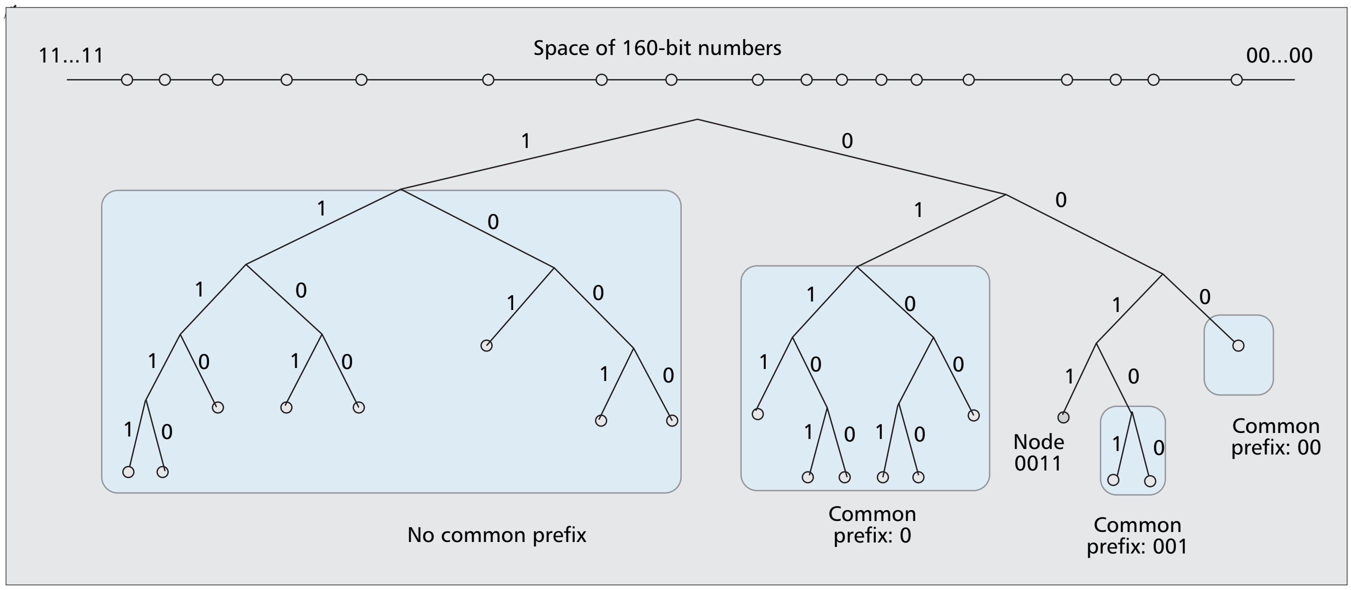
As highlighted in the Introduction, the goal driving this work is to survey existing service discovery approaches to see which ones emerge as possible candidates for a large-scale, multi-domain service-discovery infrastructure. In examining the characteristics of surveyed approaches, we have determined that to meet our goals, a service-discovery system should have the following characteristics:

- A hybrid decentralized architecture (such as the one in INS/Twine) where index nodes are provided by participating domains. This type of peer-to-peer architecture would allow different administrative domains to participate equally in the infrastructure, without requiring a central controlling entity. On the other hand, the structured components provide a backbone to the architecture, reducing maintenance and network overhead.
- Guaranteed search completeness and good correctness measures (narrowing down of results to the relevant ones). The effort in building a large-scale search architecture is wasted if it does not have the ability to discover relevant services across large network distances and domain boundaries.
- No single point of failure, and fault-tolerance in case of node failure in the distributed architecture. As a specific precaution, because of the multi-domain aspect of the



02769	X0769	XX069	XXX09	XXXX0
12769	X1769	XX169	XXX19	XXXX1
22769	X2769	XX269	XXX29	XXXX2
32769	X3769	XX369	XXX39	XXXX3
42769	X4769	XX469	XXX49	XXXX4
52769	X5769	XX569	XXX59	XXXX5
62769	X6769	XX669	XXX69	XXXX6
72769	X7769	XX769	XXX79	XXXX7
82769	X8769	XX869	XXX89	XXXX8
92769	X9769	XX969	XXX99	XXXX9

■ **Figure 20.** *Tapestry: message route path from 42769 to 31246.*



■ **Figure 21.** *Kademlia binary tree. Subtrees and corresponding prefixes are shown (boxed), w.r.t. the node with prefix 0011.*

system, the functioning of components in one domain cannot be jeopardized by the failure of those in other domains.

- Soft-state advertisement, unicast or restricted-multicast communication. The latter characteristic reduces bandwidth consumption, while the former simplifies system maintenance and improves fault-tolerance.
- A scalable, self-organizing architecture with minimal maintenance communication. As with all Internet-scale systems, a service-discovery architecture must be designed for unpredictable growth.
- Compatibility with existing security protocols. A multidomain environment benefits from simple, well-known security mechanisms.
- Platform and network-independence. This aspect allows service providers with heterogeneous platforms to participate in the architecture.
- Potential for extensibility and interoperability with other service-oriented systems. As with the scalability, the system must be flexible enough to grow and incorporate new or existing technologies.
- The use of well-accepted, standardized tools, such as Web-based technologies. In an environment where many different participants must agree on a common set of technologies, it is important for those technologies to be standardized and well-understood in the Internet community.

We have summarized the comparison of service-discovery approaches in Table 5, Table 6, and Table 7 as well as in an at-a-glance summary in Table 1, From the discussion earlier we see that none of the examined approaches achieve all the above objectives. However, it is worthwhile to examine those systems which fared well in at least some of the areas. Their desirable characteristics can then be exploited in building a unifying multi-domain, Internet-scale service-discovery system.

In selecting the components for a new service-discovery system, we first discuss the basis for the distributed peerto- peer architecture. Of the studied peer-to-peer systems, the distributed hash-table approaches (e.g. Chord) provide a desirable distributed architecture: they exhibit enough structure for low bandwidth consumption and scalability, while obviating the need for centralized components. Moreover, the deterministic query-routing algorithms of DHTs ensure query completeness and guaranteed performance (i.e. bounds on query complexity). A structured peer-to-peer system used for building a service-discovery architecture would benefit from being coupled with JXTA, which provides a platform-independent development framework, and adds standard security mechanisms.

With an architectural framework in place, we move up into the mechanism for indexing and searching service descriptions. SDS and INS/Twine both stand out as innovative, fault-tolerant approaches for query routing based on service descriptions. However, despite its clever search scheme, and a commendable focus on security, SDS relies on several centralized components which make it unsuitable for larger systems (its index, as well, strains under high advertisement load). In contrast, INS/Twine complements its DHT “undercarriage” by providing a load-balanced indexing scheme which scales well while ensuring search completeness, and eliminates irrelevant query results by an exact matching of queries to a powerful, flexible service description scheme. In combination with a soft-state advertisement mechanism, INS/Twine is a good candidate for the basis of our service-discovery scheme.

We now reach the latter design requirements that have more to do with social aspects of large-scale systems rather than the underlying mechanics. In terms of interoperability, platform independence and standardized acceptance in the Web community, the Web Service framework emerges as a clear winner. While its basic UDDI service registry approach is to a great extent based on a centralized, non-scalable architecture (and hence does not address the multi-domain aspect of the design), the set of academic efforts to expand it to a distributed architecture show that it has potential to be used in a multiple domain setting. Of all the approaches presented above, this framework has the most extensive community support, with a well-defined service description scheme that includes a platform-independent interface-description. Both the service description and the service execution protocols are open and extensible, allowing for a great degree of extensibility and the inclusion of well-accepted security mechanisms. The rich set of publicly-available standards and tools included in the Web Services framework would be a valuable tool to incorporate into a more expansive service discovery scheme, such as the one that is the eventual aim of our work.

Conclusion

In this article, we have provided an in-depth analysis of various service and resource discovery approaches. In order to provide a formal structure to our discussion and guide the reader through this document, we first defined a set of evaluation criteria that are important to any large-scale, multidomain discovery approach and outlined their related aspects. By analyzing each approach against these criteria, we brought out the strengths and weaknesses of each approach against the goal of an Internet-scale multi-domain service discovery architecture.

Based on this analysis, a comparison study of these approaches is conducted and a brief description of our findings is presented in the Technology Comparison Section. Since this survey has revealed that no single approach meets all of our defined objectives, we have focused instead on choosing those approaches that could be combined to build the desired service-discovery architecture. In our technology selection, we have focused on three particular approaches: Web Services, INS/Twine and structured P2P systems as potential components of such an architecture. They are chosen for their scalability, standardization, performance, effectiveness, security aspects, system independence and implementation support.

In the time since it was originally conducted, this survey has served as the groundwork in designing a new large-scale discovery mechanism that is multi-domain, multi-technology and aims to unite multiple diverse service-discovery platforms [79]. This platform-independent, extensible approach enables cross-technology and cross-domain service discovery, supports domain-level access control for discovery operations, and has bounded query lookup time. During the design process, the set of criteria and requirements, as well as the variety of innovative approaches that we have encountered in our survey has helped us immensely in all aspects of design and implementation.

Appendix 1 P2P Indexing Techniques

Chord [43], CAN (Content Addressable Network) [51], Tapestry [52] and Kademlia [80] are decentralized distributed indexing approaches whose only purpose is to provide structured search for associating a content to a location (node) in the network. A brief description of each of these approaches is given below.

Chord

Chord provides an efficient way for mapping a key onto a node that stores the value associated with that key. Chord is designed for peer-to-peer networks and works well even when the network is changing rapidly. In Chord, each node in the P2P network is given an m-bit ID and arranged into a logical ring in order of ascending ID. Now consistent hashing is applied to map the keys, also identified by an m-bit ID, to the nodes. To lookup a key in $O(\log N)$ time, where N is the number of nodes, each node keeps a finger table. For example a node with m-bit ID X will have $m - 1$ entries in its finger table listing nodes that will map the keys $X + 1$, $X + 2$, $X + 4$, $X + 8 \ldots X + 2^{m-1}$. It has been shown that, in the average case, keys are uniformly distributed, and the number of keys transferred due to joining or removal of node is kept small.

CAN

CAN is essentially a distributed, Internet-scale hash table that maps file names to their locations in the network. CAN supports insertion, deletion and lookup or (*Key*, *value*) pairs in the distributed hash table. CAN uses a virtual ddimensional Cartesian coordinate space (Fig. 19) to store (key *K*, value V) pairs as follows:

- First, *K* is deterministically mapped onto a point P in the coordinate space.
- The (*K*, *V*) pair is then stored at the node that owns the zone within which point *P* lies.

The *lookup* of *K* is performed by mapping *K* to *P* and requesting the corresponding node for *V*. If that node does not contain *K* then the request is routed from node to node until it reaches the node in whose zone P lies.

A new node, willing to join the CAN network, must first discover an existing CAN node. Half of the coordinate space belonging to the discovered node is assigned to the new node using the CAN routing mechanism. The new node also learns the IP addresses of its neighbors. Meanwhile the neighbors of the discovered node are notified of the new node, and update their routing tables accordingly.

Tapestry

Tapestry is a self-administering, fault-tolerant location and routing infrastructure that provides routing of messages directly to the "closest" copy of an object (or service) using only point-to-point links between nodes and without centralized resources.

Tapestry names nodes using IDs with fixed number of digits. A local routing map, or neighbor map is used at each node, to incrementally route messages to the node with the destination ID digit by digit, from the right to the left (Fig. 20). Each node therefore has a neighbor map with multiple levels, where each level represents a matching suffix up to a digit position in the ID. For example, assume a 3 digit ID. A node with ID 456 will have a routing table with entries like xx0, xx1, …, xx9 for first column, x06, x16, …, x96 for second column and 056, 156, …, 956 at third column. If the node receives a routing request for ID 236 (at k-th hop of the routing request k-digit suffix of the destination ID will be same as the ID of the routing node), it will look into the entry x36 in column two and forward the message to that node. Figure 20 illustrates the routing process of a typical request.

Although the routing information is distributed across the network the location of an object in the system is stored at a specific node, called the root node, in the system. A lookup is performed by the root node and returns the ID of the node that contains the requested object. Later, the distributed routing mechanism is used to reach the node.

Kademlia

Kademlia is a P2P distributed hash table, based on XOR metric. Like other DHT approaches, node IDs and indexes (or keys) belong to the same 160-bit space. Distance between two identifiers, say x and y, is defined as $d(x, y) = x \oplus y$. Each node is treated as a leaf in a logical binary tree. A node's position in the tree is determined by the shortest unique prefix of its ID. Each node keeps track of at least one other node in every maximal subtree that does not contain it. As shown in Fig. 21 a node with prefix 0011 keeps track of at least one node in the subtrees with prefix 1, 01, 000 and 0010 respectively. In each query routing hop an intermediate node (say A) forwards the query to another node (say B), which is closer to the search ID by at least on more bit than A; i.e. $d(B, Q) \leq 1/2d(A, Q)$, assuming Q is the query ID. This allows a node to route a query to a target node (with longest common prefix with the search ID) in Kademlia tree in $O(\log N)$ hops. The XOR-metric used in Kademlia is symmetric which allows a Kademlia node to learn (and update its routing table) from the incoming queries. This is a notable advantage of Kademlia over Chord.

SUMMARY

ADVANTAGES

P2P architectures are designed with consideration to fault-tolerance, scalability, and robustness. Structured search schemes allow the lookup process to be faster, more efficient and more specific than the broadcasting or random walk techniques adopted in partially structured and unstructured approaches. Structured search schemes also have the advantage of not requiring global knowledge of the network.

DISADVANTAGES

The Tapestry root node may be a single point of failure. Moreover, the other nodes must have prior knowledge to allow them to identify the root node. Finally, such systems assume the existence of fixed-length IDs.

REFERENCES

[1] D. Booth *et al.*, "Web Service Architecture," 2004, available: http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/

[2] F. Zhu, M. Mutka, and L. Ni, "Service Discovery in Pervasive Computing Environments," *IEEE Pervasive Computing*, vol. 4, no. 4, 2005, pp. 81–90.

[3] C. Lee and S. Helal, "Protocols for Service Discovery in Dynamic and Mobile Networks," *Int'l. J. Computer Research*, vol. 11, no. 1, 2002, pp. 1–12.

[4] G. Richard III, "Service Advertisement and Discovery: Enabling Universal Device Cooperation," *IEEE Internet Computing*, vol. 4, 2000, pp. 18–26.

[5] S. Helal, "Standard for Service Discovery and Delivery," *IEEE Pervasive Computing*, 2002, pp. 95–100.

[6] B. SIG, "Specification of the Bluetooth System — Core," http://www.bluetooth.org/docs/Bluetooth V11 Core 22Feb01.pdf, Feb. 22 2001.

[7] M. Nidd, "Service Discovery in Deapspace," Aug. 2001, pp. 39–45.

[8] S. Helal *et al.*, "Konark — A Service Discovery and Delivery Protocol for Ad-Hoc Networks," *Wireless Commun. and Net.*, vol. 3, Mar. 2003, pp. 2107–13.

[9] O. Mathieu, D. Montgomery, and S. Rose, "Empirical Measurements of Service Discovery Technologies," *IEEE Pervasive Computing*, May 2001.

[10] M. Barbeau and E. Kranakis, "Modeling and Performance Analysis of Service Discovery Strategies in Ad Hoc Networks," *Proc. Int'l. Conf. Wireless Networks*, Las Vegas, Nevada, USA, 2003, pp. 23–26.

[11] C. Bettstetter and C. Renner, "A Comparison of Service Discovery Protocols and Implementation of the Service Location Protocol," 2000, http://citeseer.nj.nec.com/article/bettstetter00 comparison.html

[12] C. Lee and S. Helal, "Context Attributes: an Approach to Enable Context-Awareness for Service Discovery," *Symp. Applications and the Internet*, Jan. 2003, pp. 22–30.

[13] G. Chen and D. Kotz, "Context-Sensitive Resource Discovery," *Proc. 1st IEEE Int'l. Conf. Pervasive Computing and Commun. (PerCom'03)*, Mar. 2003, pp. 243–52.

[14] Salutation Consortium, "Salutation Architecture Specification (part-1)," June 1999, ftp://ftp.salutation.org/salute/

[15] —, "Salutation Architecture Specification (part-2)," June 1999, available: ftp://ftp.salutation.org/salute/

[16] P. J. Leach and R. Salz, "UUIDs and GUIDs," Feb. 1998, status: INTERNET-DRAFT, hegel.ittc.ukans.edu/topics/internet/internet-drafts/draft-l/draft-leachuuids-guids-01.txt

[17] Sun Microsystems, Inc., "RFC 1057: RPC — Remote Procedure Call Protocol Specification Version 2," 1988, Available: http://www.ietf.org/rfc/rfc1057.txt

[18] Salutation Consortium, "Salutation-Lite Open Source," 2003, available: http://www.salutation.org/lite/ litesource.htm

[19] S. E. Czerwinski *et al.*, "An Architecture for a Secure Service Discovery Service," *MobiCom '99: Proc. 5th Annual ACM/IEEE Int'l. Conf. Mobile Computing and Networking*, New York, NY, USA: ACM Press, 1999, pp. 24–35.

[20] B. Bloom, "Space/Time Trade-offs in Hash Coding with Allowable Errors," Commun. ACM, vol. 13, no. 7, 1970, pp. 422–26.

[21] E. Guttman *et al.*, "RFC 2608: Service Location Protocol, Version 2," 1999, Status: PROPOSED STANDARD, Available: http://www.ietf.org/rfc/

[22] SRVLOC, "Service Location Protocol (svrloc) Working Group," 1997, http://www.ietf.org/html.charters/svrloccharter.html

[23] C. Perkins and E. Guttman, "RFC 2610: DHCP Options for Service Location Protocol," 1999, status: PROPOSED STANDARD, available: http://www.ietf.org/rfc/

[24] IANA, "IANA Protocol and Service Names," 2004, http://www.iana.org/assignments/service-names

[25] P. S. Pierre, S. Isaacson, and I. McDonald, "Printer Service Template," 2002, available: http://www.iana.org/assignments/svrloctemplates/ printer.1.0.en

[26] Sun Microsystems, "Device Architecture Specification," Sun Microsystem, Inc, Tech. Rep., June 2003, www.jini.org/nonav/standards/davis/doc/specs/html/devicearchspechtml.

[27] Sun MicroSystems, "The Davis Project," 2003, available: http://davis.jini.org/

[28] W. AdjieWinoto *et al.*, "The Design and Implementation of an Intentional Naming System," Symp. Operating Systems Principles, 1999, available: citeseer.nj.nec.com/adjiewinoto99design.html, pp. 186–201.

[29] M. Balazinska, H. Balakrishnan, and D. Karger, "INS/Twine: A Scalable Peer-to-Peer Architecture for Intentional Resource Discovery," Proc. 1st Int'l. Conf. Pervasive Computing, Springer-Verlag, 2002, pp. 195–210.

[30] UPnP Forum, "UPnP device architecture 1.0," May 2003, http://www.upnp.org/download/ UPnPDA10 20000613.htm

[31] W. W. W. C. W3C, "Extensible Markup Language (XML) 1.0 (2nd Edition)," 2000, available: http://www.w3.org/TR/2000/RECxml- 20001006

[32] Y. Y. Goland *et al.*, "Internet-draft: Simple Service Discovery Protocol/1.0," 1998, http://www.upnp.org/download/draft_cai_ssdp_v1_03.txt.

[33] R. Troll, "Automatically choosing an IP address in an Ad-Hoc IPv4 network, IETF draft," 1999.

[34] E. Christensen *et al.*, "Web Service Definition Language," 2001, available: http://www.w3.org/TR/wsdl

[35] I. Foster *et al.*, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, Open Grid Service Infrastructure WG, Global Grid Forum," 2002, available: http://www.globus.org/research/papers/ogsa.pdf

[36] UDDI Consortium, "UDDI Technical White Paper," 2002, available: http://www.uddi.org/pubs/Iru_UDDI Technical_White_Paper.pdf

[37] J. Garofalakis *et al.*, "Web Service Discovery Mechanisms: Looking for a Needle in a Haystack?" *Int'l. Wksp. Web Engineering*, 2004.

[38] P. Rompothong and T. Senivongse, "A Query Federation of UDDI Registries," *ISICT*, 2003.

[39] M. Schlosser *et al.*, "A Scalable and Ontology-Based p2p Infrastructure for Semantic Web Services," *Proc. 2nd Int'l. Conf. P2P Computing*, 2002.

[40] M. Montebello and C. Abela, "Daml Enabled Web Service and Agents in Semantic Web," *Wksp. Web, Web Services and Database Systems, LNCS*, 2003.

[41] Y. Li *et al.*, "Pwsd: A Scalable Web Service Discovery Architecture based on Peer-to-Peer Overlay Network," *Proc. APWeb, LNCS*, vol. 3007, 2004.

[42] C. Schmidt and M. A. Parashar, "Peer-to-Peer Approach to Web Service Discovery," WWW: Internet and Web Information Systems, vol. 7, 2004.

[43] I. Stoica *et al.*, "Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications," *IEEE/ACM Trans. Net.*, vol. 11, no. 1, Feb. 2003, pp. 17–32.

[44] W. W. W. C. W3C, "Xquery," 2004, http://www.w3.org/XML/Query

[45] T. Berners-Lee, R. Fielding, and L. Masinter, "RFC 2396: Uniform Resource Identifiers (URI): Generic syntax," 1998, status: PROPOSED STANDARD, ftp://ftp.internic.net/rfc/rfc2396.txt, ftp://ftp.math.utah.edu/pub/rfc/rfc2396.txt

[46] B. Traversat *et al.* Yeager, "Project JXTA 2.0 Super-Peer Virtu-

al Network," 2003, http://www.jxta.org/project/www/docs/JXTA2.0protocols1.pdf
[47] R.Rivest, "RFC 1321: The MD5 Message-Digest Algorithm," 1992.
[48] F. Zhu, M. Mutka, and L. Ni, "Splendor: A Secure, Private, and Location-Aware Service Discovery Protocol Supporting Mobile Services," *Proc. 1st IEEE Int'l. Conf. Pervasive Computing and Commun. (PerCom'03)*, Mar. 2003, http://www.cse.msu.edu/ zhufeng/splendor.pdf, pp. 235–42.
[49] M. Ripeanu and I. Foster, "Mapping the Gnutella Network: Macroscopic Properties of Large-Scale Peer-to-Peer Systems," *Proc. 1st Int'l. Wksp. PeertoPeer Systems (IPTPS '02)*, 2002.
[50] I.Clarke, O.Sandberg, and B.Wiley, "Freenet: A Distributed Anonymous Information Storage and Retrieval System," *Wksp. Design Issues in Anonymity and Unobservability*, June 2000.
[51] S. Ratnasamy *et al.*, "A Scalable Content-Addressable Network," *Proc. 2001 Conf. Applications, Technologies, Architectures, and Protocols for Computer Commun.*, ACM Press, 2001, pp. 161–72.
[52] B. Y. Zhao *et al.*, "Tapestry: A Resilient Global-Scale Overlay for Service Deployment," *IEEE JSAC*, vol. 22, no. 1, 2004, pp. 41–53.
[53] D. Meyer, "Administratively Scoped IP Multicast," RFC 2365, Internet Engineering Task Force, July 1998.
[54] C. Dabrowski, K. L. Mills, and J. Elder, "Understanding Consistency Maintenance in Service Discovery Architectures in Response to Message Loss," *Active Middleware Services*, 2002, pp. 51–60.
[55] ——, "Understanding Consistency Maintenance in Service Discovery Architectures During Communication Failure," *Wksp. Software and Performance*, 2002, pp. 168–78.
[56] C. Dabrowski and K. Mills, "Understanding Self-Healing in Service Discovery Systems," *Proc. 1st Wksp. Selfhealing Systems*, ACM Press, 2002, http://doi.acm.org/10.1145/582128.582132, pp. 15–20.
[57] S. Kent and R. Atkinson, "RFC 2406: IP Encapsulating Security Payload (ESP)," 1998, status: PROPOSED STANDARD.
[58] S. Chokhani *et al.*, "RFC 3647: Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework," 2003, status: INFORMATIONAL.
[59] T. Dierks and C. Allen, "RFC 2246: The TLS Protocol Version 1.0," 1999, status: PROPOSED STANDARD.
[60] OASIS WSS TC, "Web Services Security: SOAP Message Security 1.0," 2004, http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf
[61] —, "Web Services Security: UsernameToken Profile 1.0," 2004, http://docs.oasis-open.org/wss/2004/01/oasis- 200401-wss-username-token-profile-1.0.pdf
[62] —, "Web Services Security: X.509 Certificate Token Profile," 2004, http://docs.oasis-open.org/wss/2004/01/oasis- 200401-wss-x509-token-profile-1.0.pdf
[63] P. S. Pierre and T. Mori, "Salutation and SLP," the Salutation Consortium, http://www.salutation.org/techtalk/slp.htm
[64] IBM, "Web Service Resource Framework," 2004, http://www-106.ibm.com/developerworks/library/ws-resource/
[65] J. Allard *et al.*, " J. M. U. A. A. J. Interoperability," *Proc. 2003 Int'l. Symp. Applications and the Internet*, 2003.
[66] T. Koponen and T. Virtanen, "A Service Discovery: A Service Broker Approach," *Proc. 37th Hawaii Int'l. Conf. System Sciences*, 2004, http://csdl.computer.org/comp/proceedings/hicss/2004/2056/09/ 205690284b.pdf
[67] S. R. Livingstone, "Service Discovery In Pervasive Systems," 2003, the School of Information Technology and Electrical Engineering, University of Queensland, Australia, http://innov-expo.itee.uq.edu.au/2003/exhibits/s370816/
[68] OpenSLP, 2003, http://www.openslp.org/
[69] Sun Microsystems, "Jini Technology Starter Kit Version 2.0-002," Feb. 2004, http://wwws.sun.com/software/communitysource/ jini/download.html
[70] MIT, "INS/Twine v2," 2002, http://nms.lcs.mit.edu/software/instwine/ins-2-0.tgz
[71] Intel, Inc., "Intel Software for UPnP Technology," 2004, http://www.intel.com/technology/UPnP/download.htm
[72] SourceForge, "Linux SDK for UPnP Devices 1.2.1 (libupnp)," February 2003, http://upnp.sourceforge.net/
[73] Project JXTA, "JXTA J2SE 2.2.1," 2004, http://platform.jxta.org/java/release/2004Q1 Churrasco/release note.html
[74] —, "JXTA-C," 2004, http://jxta-c.jxta.org/
[75] IBM, "IBM WebSphere Software Platform," http://www-306.ibm.com/software/info1/websphere/index.jsp
[76] Microsoft, "Microsoft .NET framework," 2004, http://msdn.microsoft.com/netframework/
[77] Sun Microsystems, "Sun ONE Web Services," 2004.
[78] The Globus Alliance, "The Globus Toolkit," 2004, http://www-unix.globus.org/toolkit/
[79] N. Limam *et al.*, "OSDA: Open Service Discovery Architecture for Efficient Cross-Domain Service Provisioning," *Computer Commun. J.*, special issue for Emerging Middleware for Next Generation Networks, 2005, to appear.
[80] P. Maymounkov and D. Mazi, "Kademlia: A Peer-to-Peer Information System based on the XOR Metric," *Proc. IPTPS*, 2002, pp. 53–65.

BIOGRAPHIES

REAZ AHMED (r5ahmed@uwaterloo.ca) received the B.Sc. and M.Sc. degrees in Computer Science from the Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh in 2000 and 2002, respectively. He completed Ph.D. program in 2007 in Computer Science at the University of Waterloo, Canada. He has served as a reviewer for many international conferences and journals. His research interests include wide area service discovery, loosely-coupled distributed databases and content sharing peer-to-peer networks with focus on search flexibility, efficiency and robustness. He is the recipient of the Canadian Commonwealth Scholarship, University of Waterloo Graduate Scholarship, and Merit scholarship and Dean's award in BUET.

NOURA LIMAM (noura.limam@lip6.fr) received the B.S. degree from the National School of Computer Science, Tunisia, in 2001, and M.S. degree in networking from the University of Paris VI, France, in 2002. In 2003, she was with Ucopia Communications Inc., France, at the R&D Department where she was involved in the development of a management tool for enterprise wireless networks. From 2004 to 2005 she was a research assistant at the School of Computer Science at the University of Waterloo, Canada. She is currently working towards the Ph.D. degree at the University of Paris VI. Her research interests include network and service management, service discovery and service-oriented architectures. Jin Xiao (j2xiao@bbcr.uwaterloo.ca) received his B.Sc. first class honors in Computer Science from the University of Calgary, Canada in 2001. He is currently a Ph.D. candidate at the School of Computer Science at University of Waterloo, Canada. He conducts research in the areas of network and distributed systems management, economic modeling, network service quality assurance, and autonomous management system design.

YOUSSEF IRAQI (y iraqi@du.edu.om) received M.S. and Ph.D. degrees in computer science from the University of Montreal, Canada, in 2000 and 2003, respectively. He is currently an Assistant Professor and Chairman, Department of Computer Science, Dhofar University, Sultanate of Oman. From 2003 to 2005 he was a research assistant professor at the School of Computer Science at the University of Waterloo, Canada, and from 1996 to 1998 he was a research assistant at the Computer Science Research Institute of Montreal. His research interests include network and distributed systems management, resource management in multimedia wired and wireless networks, and peer-to-peer networking.

RAOUF BOUTABA (rboutaba@uwaterloo.ca) is a Professor of Computer Science at the University of Waterloo, Canada. His research interests include network, resource and service management in wired and wireless networks. He is the founder and Editor-in-Chief of the IEEE Transactions on Network and Service Management and on the editorial boards of several other journals. He is a distinguished lecturer of the IEEE Communications Society, the chairman of the IEEE Technical Committee on Information Infrastructure and the IFIP Working Group 6.6 on Network and Distributed Systems Management. He has received several best paper awards and other recognitions such as the Premiers research excellence award.