Classification of Service Discovery in Pervasive Computing Environments

Feng Zhu, Matt Mutka, and Lionel Ni

Michigan State University, East Lansing

Abstract: Service discovery is an essential task in pervasive computing environments. Simple and efficient service discovery enables heterogeneous and ubiquitous computing devices and services to be easier to use. Service integration uses services as building blocks to achieve complex services. We describe service discovery and service integration, analyze design issues, and categorize the service discovery protocols.

Keywords: Service Discovery, Service Integration, Service Security.

In 1991, Mark Weiser coined the term *ubiquitous computing*, which is also called *pervasive computing*. In pervasive computing environments, people are surrounded by a variety of computing devices. Those devices communicate with each other and provide information "at a glance" without our "active attention" [1]. Presently, PCs, notebooks, cell phones, and Personal Data Assistants (PDAs) surround us. In the near future, additional networked computers, ranging from tiny sensors to extremely dynamic and powerful devices will provide a variety of information and services. It becomes overwhelming to manage these devices, configure different kinds of applications, and dynamically find the available computing services in such pervasive computing environments.

Service discovery protocols enable computers to be easier to use. They facilitate interaction between computers, with an aim to approach zero administration overhead and therefore free users from tedious and redundant administrative and configuration work. Therefore, service discovery research is critical to the success of pervasive computing [2].

Let's consider the basic problem of service discovery in this scenario. Bob visits Michigan State University (MSU). He turns on his PDA. We call the PDA a *client*. He searches the Web and finds an online MSU map. The map, however, is too small to view on his PDA. He enters the nearby Engineering Building (EB) to try to find a printer for printing the MSU map. The PDA begins locating a printer. Many printers in the EB provide printing *services*. Nevertheless, Bob is only allowed to use some of the public printers as a guest. After finding two printers, Bob picks the one that is closer to him. Then the PDA contacts the printer. Although the PDA may not have a driver for the printer, it may still determine how to use the printer without Bob's intervention. Then, Bob prints the map successfully. In this scenario, Bob explicitly uses the printing service. He also implicitly uses the wireless connection service available on the campus and the location information service to help him find a nearby printer.

SERVICE DISCOVERY MODELS

A trivial service discovery model is one in which clients know the services in advance, or clients have already cached the services' information, so that clients do local lookups before contacting the services. In simple environments such as home environments, clients inquire about services first, and then contact the services. To support thousands of computing services, such as the services in the EB at MSU, we may optionally use *directories* to store all the service information. A client queries a directory for service information and then contacts services. Nevertheless, adding directories introduces a level of indirection. Next, we illustrate what a client does from its point of view. In turn, we show the views of a service and a directory.

From the Client's Point of View

In most cases, a client is a program that runs on behalf of a user and interacts with the user. We list the client's steps below.

- A client queries directories for services. A client either browses services or looks for a specific service.
- Alternatively, without going through directories, a client directly queries all the services. All the services that meet the query requirement reply back to the client.
- Then the client program or the user selects a service to use.
- Finally, the client uses the service.

With service discovery software installed, the client does not need to configure server settings, such as printer servers. No drivers need to be pre-installed. The next time, when that printer is replaced, users can use the new printer without worrying about installing a new driver. Furthermore, computing system support staff will be released from the burden of upgrading and installing software on all client machines. In addition, service discovery provides fault tolerance transparently.

From the Service's Point of View

A service has a name, a list of attributes, and user privileges. For instance, a printer says it provides printing service and it is able to provide color printing at 720 by 720 dpi. It might only allow people in the marketing department to use it. When a service needs to use other services, we call it a client. Services work as follows.

- A service announces its information to clients or directories. For example, every ten minutes a printer announces its information to let clients or directories know its existence.
- Alternatively, a service answers directory solicitation or client queries.

- A service authenticates and authorizes the user when a client asks for service.
- Finally, the client is granted service access and uses the service through the service's interface.

From the Directory's Point of View

With directories available, a client queries twice, the first time asking directories and the second time contacting the service(s). Without directories, a client looks for services directly.

- On hearing a service announcement, directories first check privileges; and then service information will be updated or recorded in the directories.
- Alternatively, directories may ask what services are available instead of waiting for the service to be announced.
- When receiving a query from a client, directories authenticate, authorize, and reply to the client.

AN ANALYSIS OF SERVICE DISCOVERY DESIGNS

Much active service discovery research has been occurring. We discuss major service discovery products or projects in the sidebar. Targeted at different environments, these service discovery protocols have different design criteria and choices. We categorize these designs and then compare and contrast them.

Service Designs

We describe a service as some computing resource used by users, user programs, or other services. Using our scenario as an example, printing services, location information, and wireless network connections are services.

Service Naming. A service has a name. Suppose Bob uses a printer. Printing is the name of the service that Bob uses. Nevertheless, the problem is when Bob looks for a printing service, a printer calls itself a print service. Then Bob is unable to find the printer [2]. Most protocols solve this problem by defining a service naming standard, which avoids the naming conflict. Bluetooth (see Appendix) maps service names to 128-bit numbers. Defining services in SLP (see Appendix) should follow a service template [3].

It is likely that many service discovery protocols co-exist. When a mobile client moves from one service discovery domain to another service discovery domain, the mobile client needs to understand different service protocols and use different vocabularies, for example saying print service at one time and printing service at another time.

The other problem is how to support new services. Although it is easy to add a new service name to a service protocol standard, it is difficult for users and client programs to know it automatically. Very likely, users need to browse for service names and then learn the new terms.

Service Attributes. A service usually has many attributes. To avoid conflicts, service attributes also have standard naming conventions as service names. A client's request is matched against services' attributes. When a client supplies more precise query requirements, fewer services will be selected. As a result, less network traffic is generated and fewer services are involved. If a query is too strict, no services may be matched and then the client needs to query again with fewer constraints. In addition to search functionality, almost all service discovery protocols provide wild card searches, which let clients examine all the available services.

Service Invocation. After discovering the service, a client invokes the service through a service interface. Some protocols such as the Bluetooth Service Discovery Protocol leave the service interface for applications to define. Some protocols base on Remote Procedure Call (PRC). Salutation (see Appendix) is such an example. Some protocols use downloadable code. For example, Jini (see Appendix) uses downloadable Java code. Other service protocols only transfer data. UPnP (see Appendix) achieves service invocation based on eXtensible Markup Language (XML), Simple Object Access Protocol (SOAP), and Hyper Text Transform Protocol (HTTP).

Service invocations in Jini and UPnP need TCP/IP protocols, HTTP servers, or Java Virtual Machines (JVMs), which may not suitable for very resource limited devices. Special design considerations are needed for those wireless devices that have limited network bandwidth and power.

Service Status Inquiry. A client may be interested in services' events or status changes. One way of knowing about them is by polling the service. Another way, known as service event notification, is by registering with the service and the service will notify clients who have shown interest. Most protocols implement service event notifications. If events are generated very frequently or a service status changes very fast, it is better to use service polling.

It is even better to have agents do event filtering and aggregation. Jini provides several such methods. Services send events to agents and let agents make sure all the events are delivered to clients; an agent may act as a sink for events, which will be filtered, aggregated, and then sent to clients; or an agent may also resemble a mailbox to filter events over time. Although clients and services benefit from event filtering and aggregation, some computers in networks need to provide computing resources to handle the events. We compare different service discovery product designs in Table 1.

Table 1. Comparison of service discovery protocols.

	Bluetooth	DEAPspace	INS	Jini	Salutation	SLP	SSDS	UPnP
Service[1]								
Naming and attributes	Standard	N/A	N/A	Standard	Standard	Standard	N/A	Standard
Invocation	N/A	N/A	N/A	Java code	Remote Procedure Call	URL	N/A	XML Data
Status inquiry	N/A	N/A	N/A	Notification and event agent	Notification	N/A	N/A	Polling and notification
Directory								
Centralized vs. distributed	N/A	N/A	Distributed	Distributed	Either	Centralized	Distributed	N/A
Number of Service Information Copies	N/A	N/A	Fully replicated in sub domains, single copy globally	Multiple copies	Multiple copies	Multiple copies	Single copy	N/A
Flat vs. hierarchical	N/A	N/A	Flat and hierarchical [2]	Flat or hierarchical	Flat	N/A	Hierarchical	N/A
Service State in Directories	N/A	N/A	Soft state and hard state [3]	Soft state	Hard state with periodically check	Soft state	Soft state and hard state [3]	N/A
Directory address	N/A	N/A	Configured address	Configured or multicast address	Configured or multicast address	Multicast address	Multicast address	N/A
Number of Directory Hierarchies	N/A	N/A	Multiple hierarchies	Single hierarchy	N/A	N/A	Multiple hierarchies	N/A
Announcement and lookup								
Query vs. announcement	Query	Announcement	Both	Both	Both	Both	Both	Both
Directory-based vs. non-directory-based	Non-directory-based	Non-directory-based	Directory-based	Directory-based	Directory-based	Either	Directory-based	Non-directory-based
Communication	Unicast and broadcast	Broadcast	Unicast, anycast, and multicast	Unicast and multicast	Unicast and broadcast	Multicast	Unicast, multicast, and broadcast	Unicast and multicast
Service Selection								
User vs. Protocol Selection	User selection	User selection	Protocol selection	User selection	User selection	User selection	User selection	User selection
Service Matching	Match all	N/A	Match best	Match all	Match one or match all	Match all	Match all or match best	Match all
Context-aware	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Scope-aware	Location /vicinity	N/A	Location and administrative domain	Location and administrative domain	N/A	Administrative domain	Location, administrative domain, and network topology	N/A
QoS-aware	N/A	N/A	Yes	N/A	N/A	N/A	N/A	N/A

[1] We do not compare service design of the research institute projects, since they do not have or publish their standards.
[2] INS: Flat in sub-domain and hierarchical globally; INS/Twine: flat, peer-to-peer.
[3] Soft state at leaf directories and hard state at other directories.

Directory Designs

Directories cache service information and answer clients' lookup requests. Thus, the overhead of handling unrelated requests for services and the communication between clients and unrelated services are removed. More importantly, this facilitates large-scale service discovery. Directory architectures, service information cache strategies, and hierarchies are different depending on the environments.

Centralized vs. Distributed Directories. A centralized directory stores all the services' information in a central location. The directory is likely to be a bottleneck and the single point of failure, which causes the whole system's failure. In large service discovery domains, it is inefficient to go through a centralized directory all the time.

Most of the service discovery protocols use distributed directories, which store services' information within their own domains. Service information is distributed among directories. A directory failure only affects part of the system. With less information in each directory, service lookup within a directory is more efficient. On the other hand, a service lookup may go through several directories. In contrast, a service lookup in a centralized directory only goes to one directory with less network communication overhead and latency.

Number of Service Information Copies. For each service, the service information may be a single copy, multiple copies, or fully replicated in directories. Many protocols have a single copy of services in its domain. A directory failure will affect the domain for which it is responsible. In Jini and SLP service discovery environments, multiple directories may coexist. Therefore, multiple copies of service information may exist. It is more reliable with multiple copies of service information in several directories, but the greater the number of directories, the greater the overhead. INS (see Appendix) implements fully replicated copies within a sub domain. The advantage of fully replicated directories is that a service search only goes to the directory to which a client is attached. Multiple copies or fully replicated copies of service information should be consistent in directories. Otherwise, querying different directories may result in different service information and may cause problems.

Flat vs. Hierarchical Directory Structure. In a flat directory structure, directories have peer-to-peer relationships. In one type of flat directory structure, directories connect to each other and exchange information. In INS, directories have a flat mesh structure within its domain and exchange information with all other directories, so that all service information is available locally and the service search is very efficient. These information exchanges generate much communication traffic, and therefore it is not scalable. INS/Twine based on peer-to-peer technology, as discussed in the sidebar, is much more scalable because service information is not replicated.

While in a hierarchical directory structure, directories have parent and child relationships. Domain Name System (DNS) is an example of a hierarchical directory structure. Searching through the directory hierarchy is necessary. For example, a service discovery protocol is based on widely available DNS servers to do service discovery [4]. Many other service discovery protocols also use tree-like hierarchical structures to provide scalable solutions. Nevertheless, it is difficult to make directories both scalable and efficient.

Service State in Directories. Most service discovery protocols maintain service status as soft states. In a service announcement, the life span of the service is specified. Before the service expires, it should announce itself again to renew the service. Otherwise, the service will not be valid to use after expiration. In the mean time, expired service entries will be wiped off from the directories. In case a service is down, that service will not be available after its lifespan and clients will not use it. Therefore, directories are free from monitoring service states. This elegant soft state service management mechanism greatly simplifies the system design and makes service information fresh. On the other hand, regular service announcements require more network bandwidth, and put extra load on the directories.

On the contrary, directories may maintain service status as hard state. In this case, the directories keep the service status until it is told to change the service status information. Using hard state directories, few service announcements and housekeeping jobs are required. On the other hand, it is difficult to guarantee all service information is up to date in hard state directories. Services may go down without notifying directories or out of date service status results from network communication error. One solution is that directories can poll services for status periodically.

Directory address. Conventional directories, such as DNS, listen on well-known ports and clients are manually configured with the directory addresses. Moving to different networks, clients need different directory addresses. With DHCP servers, manual configuration is not necessary, but DHCP servers need to be deployed.

To avoid directory address configuration or support from DHCP servers, some service discovery protocols use multicast addresses. Directories listen to and talk on a multicast address, such as in SLP and SSDS. Clients query for services by sending requests to a multicast address or by listening on a multicast address. If multiple directories reply, clients can pick one directory to further communicate. Using multicast addresses, directories also provide fault tolerance features. A directory failure will not cause any problems, since other directories listening to the same address may provide the same services.

Number of Directory Hierarchies. A single hierarchy directory has a tree structure, while a multiple hierarchy structure could be a forest or many trees sharing a set of leaf directories. Multiple hierarchies index service information on different keys. Like a database index, service information search based on a

key may greatly speed up the search. Extra computing resources and house keeping jobs are obvious overhead for multiple hierarchy directories.

Many service discovery protocols implement more than directory functionalities. Directories in INS not only cache service information, but also select the best services for users. In Table 1, we compare those service discovery protocols, which have directories.

Service Announcement and Lookup

Service announcement and lookup are the key parts of service discovery protocols. Query and announcement are the two basic mechanisms for clients, services, and directories to exchange information. Service announcement and lookup may also be categorized into non-directory-based and directory-based approaches. Furthermore, four different communication techniques are used in service discovery protocols: unicast, anycast, multicast, and broadcast. Based on the OSI reference model, these four communication techniques may be at the data link layer (media access control sub-layer), at the network layer, or at the application layer.

Query vs. Announcement. The two methods for clients to learn which services are available are query and announcement, also known as active and passive or pull and push. As announcements go to all the clients or directories, interested clients or directories do not need to ask separately for the same service. Nevertheless, clients or directories have to handle all the announcements, regardless of whether they are interested. When asking actively, a client or a directory will receive an immediate response. While listening to service announcements, a client or a directory may wait up to the interval of service announcement.

Unicast. Unicast is widely used in many service discovery protocols. When a client knows a directory's network address in advance, it will send a unicast message to the directory. If a client knows a service provider's address, it will contact the service provider directly. Furthermore, if a service's address is known to a directory or vice versa, service announcements and queries between a directory and a service are also using unicast.

Anycast. A set of similar services all may meet a client's request. The service request sent to one of the set of services is known as anycast. For instance, INS uses overlay network anycast, so its anycast is at the application layer. In INS, a client's request goes to a directory. After searching that directory, INS routes the request to a service based on the application-defined service weight. Thus, a client request goes to a service with the best service weight.

Multicast. The drawback of unicast is that the network address needs to be configured or known ahead of time. On the contrary, in many situations, the addresses are unknown. A solution is that clients, service, and directories use multicast addresses for announcements and queries. For example, SLP uses

TCP/IP network layer multicast addresses. It is simpler for mobile clients and will be automatically compatible when new services or directories are added later, because global multicast addresses are used. Nevertheless, there are very limited globally multicast addresses at the network layer. Moreover, multicast is not allowed on some routers even though the routers have multicast capabilities. Using multicast also introduces more communication overhead compared to unicast, since more nodes are involved in the communication.

Broadcast. Sometimes broadcast is used in service discovery protocols. For example, Bluetooth Service Discovery Protocol uses broadcast to find other services. In Bluetooth, as other communication techniques are based on broadcast, it is simpler to use broadcast directly. Another example is that Salutation can utilize broadcast if underlying protocols support broadcast. Regardless, data link layer broadcast is usually limited to its subnet.

Using unicast usually saves much communication traffic; using anycast simplifies client side processing; using multicast saves administrative overhead; and using broadcast is sometimes more efficient. We compare service announcements and lookups in Table 1.

Service Selection

While many similar services are available to a client, which service should the client use? It is challenging to find services for users efficiently and accurately.

User vs. Protocol Selection. Service discovery protocols may select services for a user. In INS, for example, the protocol decides which service a client should use. For most service discovery protocols, client programs or users choose from a matched list of services. The advantage of protocol selection is that it simplifies client programs or little user involvement is needed. On the other hand, protocol selection may not reflect the actual user's will. Predefined selection criteria may not apply to all cases. Alternatively, too much user involvement causes inconvenience. For example, it may be tedious for a user to examine many printers and compare them. A balance between protocol selection and user selection is preferred.

Service Matching. Some service discovery protocols match one of the services for a client. In Matchmaking, a classified advertisement matchmaking framework, client requests are matched to services and one of the matched services is selected for user [5]. In INS, the service discovery protocol matches the best service based on application defined metrics. Most protocols match all the services and let the user choose.

Context-awareness. Context information is useful in selecting services. For example, when Bob drives on the highway, his cell phone uses a Bluetooth connection to find his earphone. While he wants to access his email, his cell phone uses a 3G connection. In the above two scenarios, selecting one of the

connections to use for the cell phone is based on context information. Either intelligence should be built in his cell phone or user involvement is necessary for better service discoveries. So far, only a few projects use location information as a kind of context information to help service selection. We discuss more about location-awareness in the next paragraph.

Scope-awareness. To support a large amount of services, defining and grouping services in scopes facilitates service search. Location-awareness is a key feature in pervasive computing [1] and location information is helpful in many service discovery cases.

Although location information is very important and there has been much location awareness research for indoor and outdoor location sensing techniques, few service discovery protocols integrate location information. Research at MIT [6] integrates Cricket into INS to provide location dependent service discovery. Another example is Jini, in which location information is an optional attribute for services. We say this is a scope searched by physical location.

The administrative domain is another kind of scope, which is supported by many protocols. For example, while Jack is at home and looking for a printer, he should not find Bob's printer in Bob's house although it is next door. Jack's domain and Bob's domain are two separate domains. In an enterprise environment, we see more examples of administrative domains.

When Bob wants to watch news online on his PDA with a wireless LAN connection, he needs software to cache the video data for his PDA in case the wireless connection between his PDA and the wireless access point is intermittently faulty. That caching software is preferred to be close to his PDA, for example, within one hop of the network. This is a scope search by network topology. In SSDS, similar scopes are mentioned.

These geographical location information, administrative domain information, and network topology information may be attributes of services. Multiple hierarchies may be built based on these categories; so different service searches may utilize different hierarchies. It is worthwhile to build different directory hierarchies just as phone books have yellow pages and white pages.

Much research has proposed locating objects in a wide area. Some of them use a single directory hierarchy, and others use multiple directory hierarchies. No matter if there is a single hierarchy or multiple hierarchies, the difficult problem is how to express the service information at different levels of the hierarchies.

First, what services need to be listed in upper level directories? Second, what service information to store in lower level directories and what service information to store in higher level directories? To avoid being a bottleneck, upper level hierarchy directories should be concise. Filtering and aggregating service information is necessary when building the upper level hierarchies. Third, updating service information in the upper level hierarchies may overwhelm the directories, when many services update information at

the same time. Service status changes and mobile services moving all cause the directories to be busy updating.

In SSDS, service information in non-leaf level directories is hashed and filtered into indexed bit numbers. High compression is achieved. Nevertheless, the directories need to be built again and again over time, since the algorithm is not able to remove stale services. Another example of locating mobile objects in a wide area is Globe [7]. In the Globe architecture, an indirection layer of name to address mapping is added to handle mobile objects gracefully. Object names are independent of addresses. Name to address mapping happens when an object is searched. Castro, et al. proposed a protocol to access specific application data across different service discovery domains with different service discovery protocols [8].

QoS-awareness. Providing users with better services and balancing services usage are nice features for service discovery protocols. For better service matching, service requests may be directed to less loaded services or better resource price ratio services.

Service attributes are defined to match client requests more precisely. Nevertheless, most protocols only support static attributes. Let's use a printer's printing speed as an example; the current load of a printer is dynamic and should be taken into consideration. If a printer announces its current load as an attribute, it will announce its service more frequently than if it has only static information. Much more communication traffic is generated and directories are more busy handling announcements. To reduce the directory's update and network overhead, printers may wait for clients to query. Clients, however, may spend too much time determining from which printer to print.

At the service side, sharing the loads and balancing them on different services is also preferred. Few protocols define application metrics-based load balancing. A good example is INS. Applications define their metrics and service lookups are based on the metrics. In Table 1, we compare service selection of different service discovery protocols.

Security and Privacy

We consider user authentication, service authorization, confidentiality, integrity, non-repudiation, availability, and user privacy in service discovery protocols [9]. Although there is much research related to service discovery, few protocols have full security and privacy functionality built-in.

User Authentication and Service Authorization. Protecting services from unauthorized use is necessary. For example, we do not want a storage service to be accessible by anyone. Furthermore, users of a storage service should not be allowed to access other's files arbitrarily. The problem is that it is not realistic for each service to maintain its users and an access control list. User passwords may be different on different services. Small devices may be overburdened to handle authentication and authorization.

In many network infrastructures, servers are available to store user information and distribute cryptographic keys. With these servers, providing authentication and authorization is not difficult. While it is more difficult to obtain these servers in ad hoc environments, it is even worse for some devices with very limited processing and communication capability to do authentication and authorization.

Confidentiality and Integrity. Confidentiality and integrity in service discovery are primarily communication security. Communication between service discovery components should be safe. Malicious users may listen to communication channels or even actively attack systems. We do not want service information exposed to malicious users or changed during communication. These requirements are translated to use of message encryption and message authentication code.

Availability, User Privacy, and Non-repudiation. Services and directories may be targets of attackers. Making services and directories available against attack is similar to other network applications. User privacy is always a concern. We want to use services easily but keep our information private. In the mean time, service usage should be tracked. A digital signature is usually used to achieve non-repudiation of the service usage.

Deploying security in service discovery protocols means more administrative overhead. Proper permissions need to be set for services and users. With thousands of services and hundreds of users in an enterprise, groups or roles need to be created and privileges need to be assigned. In dynamic environments, daily administrative tasks may be overwhelming. Even if service discovery protocols try to make service usage easier, overwhelming security administrative tasks may offset some advantages. SSDS implements most of these security properties. Bluetooth has its built-in challenge-response authentication, authorization, and also an encrypted mode of communication.

AN ANALYSIS OF SERVICE INTEGRATION DESIGN

Services provide different functionalities. Taking services as building blocks, service integration can build complex and very powerful services. Service integration is also known as service composition.

Simple Service Integration vs. Complex Service Integration

Simple service integration chains services together. One service's output is another service's input. For example, Bob is driving on the highway; he wants to find the nearest McDonald's. An Internet map service's output (a driving direction map) goes to an adapter (software) to change to an output format that will fit Bob's PDA and then displays on it. Ninja Paths is an example of simple service integration [10]. To get a composite service, a path is created. Along that path, services are dynamically selected and connected.

Complex service integration may provide more complicated services. For example, Bob even wants to hear the driving directions. Then not only the graph (map) is sent to Bob, but text directions as well. Another adapter (text to speech software) converts the instructions and output to Bob's earphone. Based on the feedback of Bob's PDA about the network connection, which is sometimes good and sometimes bad, the adapters change the output format accordingly. Two complex service integration methods are discussed in [11]. One way is to create a service interface to interact with multiple services. Another way is to compose services and build a stand-alone service. Those integrated services may be used as service components to build other services.

Static vs. Dynamic Service Integration

Static service integration integrates services before a client uses the services. If one of the services fails, service integration needs to start over again. Dynamic service integration may replace failed services or add in more services if necessary without starting over the service integration processes. Dynamic service integration is more difficult to implement than static service integration since every service component of the dynamic service is being monitored and should be replaced immediately in case of failure.

Fault Tolerance and Failure Recovery

In dynamic and distributed environments, fault tolerance and failure recovery for service integration are two difficult issues. In the Ninja Architecture, services are monitored and paths are dynamically changed to guarantee optimal services [10]. Another example is a service integration architecture based on software hot-swapping technology proposed by Mennie and Pagurek [11]. Dabrowski, et al. modeled and analyzed different failure recovery strategies in [12]. Jini and UPnP were the two protocols that they tested. Performance responsiveness, effectiveness and efficiency were explored in that work.

In this paper, we discussed elements of service discovery protocols and their design issues. Classifications of the service discovery protocols were given. Different protocol designs were compared. In addition, we described service integration and related issues.

Appendix

Service Discovery Protocols

There are a few surveys of service discovery protocols. A good example is by Richard [13]. All the service discovery protocols in this sidebar operate in ways that are similar to what we describe in the main text in the Service Discovery Models section, with the exception of DEAPspace, which has a unique method of service announcement and lookup.

Bluetooth Service Discovery Protocol

Bluetooth, from the Bluetooth Special Interest Group (SIG), enables nearby devices to communicate with each other at low cost and low power consumption [14]. Part of the Bluetooth specification is the Bluetooth Service Discovery Protocol, which enables Bluetooth devices to discover each other.

DEAPspace

IBM Research has studied and proposed a service discovery protocol for single-hop ad hoc environments, known as DEAPspace [15]. In contrast to other service discovery protocols in which services announce their information, DEAPspace's algorithm caches service information at each node, then each node broadcasts its knowledge of other services and its own services in turn. The nodes learn from others. Service lookup is accomplished by searching the local cache. Furthermore, energy weak devices use idle mode to save power.

Intentional Naming System (INS) and INS/Twine

Researchers at MIT designed INS [16]. As services in INS are not mapped to fixed service locations, a level of indirection is added. INS resolves a service lookup to a service location at the delivery time, known as *late binding*.

In INS/Twine [17], service attributes are hashed and stored in mesh structure directories. Service lookup is based on peer-to-peer technology, a more scalable approach to handle millions of services. Service lookups, however, may go through several directories, which may have additional search latency.

Jini Network Technology

Sun Microsystems' Jini is based on Java technology [18]. One special feature of Jini is the mobile Java codes, which may be moved among clients, services, and directories. The advantage of Jini is its

platform independency, but the disadvantage is that all the clients, services, and directories depend on Java runtime environments directly or indirectly.

Salutation

The Salutation Consortium has rolled out the Salutation protocol [19]. It is an open source protocol and royalty-free. One advantage of the protocol is that it implements two interfaces. One interface is for applications. The other interface is designed to be independent of the transport layer, so that it is very flexible to use various underlying transport protocols and may be used for more environments. Furthermore, a mapping of Salutation over the Bluetooth Service Discovery Protocol has been specified [20].

Secure Service Discovery Service (SSDS)

SSDS at UC Berkeley puts emphasis on security and supporting a huge number of services, known as wide-area support [21]. Public key and symmetric key encryption are used for communication privacy and security; a Message Authentication Code (MAC) is used to ensure message integrity; and authentication and authorization are available. For wide-area support, different hierarchical directory structures are considered. A technology based on the Bloom filter [22] is used to achieve service information aggregation and filtering when building up the hierarchical directories.

Service Location Protocol (SLP) Version 2

Posted by IETF as a standard track protocol, SLP is for enterprise environments [23]. As the protocol name states, SLP only defines a way to locate a service and leaves open the interaction between clients and services after service discovery. URLs are used for service locations.

Universal Plug and Play (UPnP)

UPnP is from the UPnP Forum [24]. The major player is the Microsoft Corporation. UPnP targets unmanaged networking environments, such as home environments. UPnP is a device oriented service discovery protocol. All the service information and communication is in the XML format, which is platform and programming language independent and greatly increases interoperability between devices.

Reference

[1] M. Weiser, "The Computer for the 21st Century," *Scientific American*, vol. 265, Issue 3, 1991, pp. 66-75.

[2] T. Kindberg and A. Fox, "System Software for Ubiquitous Computing," *IEEE Pervasive Computing*, January-March, 2002, pp. 70-81.

[3] E. Guttman, C. Perkins, and J. Kempf, "Service Templates and Service: Schemes," IETF, RFC 2609, June, 1999, available at http://www.faqs.org/rfcs/rfc2609.html.

[4] S. Cheshire, "Discovering Named Instances of Abstract Services using DNS," IETF, Internet Draft draft-cheshire-dnsext-nias-00.txt, July 13, 2001, available at http://files.dns-sd.org/draft-cheshire-dnsext-nias-00.txt.

[5] R. Raman, M. Livny, and M. Solomon, "Matchmaking: Distributed Resource Management for High Throughput Computing," *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing*, Chicago, IL, 1998.

[6] A. Chakraborty, *A Distributed Architecture for Mobile, Location-Dependent Applications*, master's thesis, Engineering in Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 2000.

[7] M. v. Steen, F. J. Hauck, P. Homburg, and A. S. Tanenbaum, "Locating Objects in Wide-Area Systems," *IEEE Communications Magazine*, January, 1998, pp. 104-109.

[8] P. Castro, B. Greenstein, R. Muntz, C. Bisdikian, P. Kermani, and M. Papadopouli, "Locating Application Data Across Service Discovery Domains," *ACM SIGMOBILE MOBICOM 2001, 7th Annual Int. Conf. Mobile Computing and Networking*, Rome, Italy, 2001.

[9] W. Stallings, *Cryptography and Network Security: Principles and Practice*, 2nd ed, Prentice Hall, 1998.

[10] S. D. Gribble, M. Welsh, R. v. Behren, E. A. Brewer, D. Culler, N. Borisov, S. Czerwinski, R. Gummadi, J. Hill, A. Joseph, R. Katz, M. Mao, S. Ross, and B. Zhao, "The Ninja Architecture for Robust Internet-Scale Systems and Services," *IEEE Computer Networks*, vol. 35, Issue 4, 2001.

[11] D. Mennie and B. Pagurek, "An Architecture to Support Dynamic Composition of Service Components," *5th International Workshop on Component-Oriented Programming, WCOP 2000*, Cannes, France, 2000.

[12] C. Dabrowski, K. Mills, and J. Elder, "Understanding Consistency Maintenance in Service Discovery Architectures during Communication Failure," *Proceedings of the 4th International Workshop on Active Middleware Services*, Edinburgh, UK, 2002.

[13] G. G. Richard III, "Service Advertisement and Discovery: Enabling Universal Device Cooperation," *IEEE Internet Computing*, September-October, 2000, pp. 18-26.

[14] "Specification of the Bluetooth System -- Core," Bluetooth SIG, Version 1.1, February 22, 2001, available at http://www.bluetooth.org/docs/Bluetooth_V11_Core_22Feb01.pdf.

[15] M. Nidd, "Service Discovery in DEAPspace," *IEEE Personal Communications*, August, 2001, pp. 39-45.

[16] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley, "The design and implementation of an intentional naming system," *17th ACM Symposium on Operating Systems Principles (SOSP '99)*, Kiawah Island, SC, 1999.

[17] M. Balazinska, H. Balakrishnan, and D. Karger, "INS/Twine: A Scalable Peer-to-Peer Architecture for Intentional Resource Discovery," *Pervasive 2002 - International Conference on Pervasive Computing*,Springer-Verlag, Zurich, Switzerland, 2002.

[18] "Jini™Technology Core Platform Specification," Sun Microsystems, Version 1.2, December, 2001, available at http://wwws.sun.com/software/jini/specs/.

[19] "Salutation Architecture Specification," Salutation Consortium, Version 2.0c, June 1, 1999, available at ftp://ftp.salutation.org/salute/sa20e1a21.ps.

[20] "Mapping Salutation Architecture APIs to Bluetooth Service Discovery Layer," B. Miller, Version 1.0, July 1, 99, available at http://www.salutation.org/whitepaper/BtoothMapping.PDF.
[21] S. Czerwinski, B. Y. Zhao, T. Hodes, A. Joseph, and R. Katz, "An Architecture for a Secure Service Discovery Service," *Fifth Annual International Conference on Mobile Computing and Networks (MobiCom '99)*, Seattle, WA, 1999, pp. 24-35.
[22] B. Bloom, "Space/Time Trade-offs in Hash Coding with Allowable Errors," *Communications of ACM*, 13(7), 1970, pp. 422-426.
[23] E. Guttman, C. Perkins, J. Veizades, and M. Day, "Service Location Protocol, Version 2," IETF, RFC2608, June 1999, available at http://www.ietf.org/rfc/rfc2608.txt.
[24] B. A. Miller, T. Nixon, C. Tai, and M. D. Wood, "Home Networking with Universal Plug and PIay," *IEEE Communications Magazine*, December, 2001, pp. 104-109.