Service Discovery in Pervasive Computing Environments

Pervasive computing environments pose unique service discovery challenges. A survey and taxonomy of existing discovery protocols clarifies the open research problems for anytime, anywhere computing.

Feng Zhu and Matt W. Mutka
Michigan State University

Lionel M. Ni
Hong Kong University of Science and Technology

Pervasive computing environments gracefully integrate networked computing devices—from tiny sensors to extremely dynamic and powerful devices—with people and their ambient environments. A room, for example, might be saturated with hundreds of devices that provide information to people without needing their active attention. Service discovery is essential to achieving such sophistication. It enables devices and services to properly discover, configure, and communicate with each other. Unfortunately, we now spend precious time actively looking for services and manually configuring devices and programs. Sometimes the configuration requires special skills that have nothing to do with the tasks we want to accomplish.

Service discovery protocols are designed to minimize administrative overhead and increase usability. They can also save pervasive system designers from trying to foresee and code all possible interactions and states among devices and programs at design time. By adding a layer of indirection, service discovery protocols simplify pervasive system design.

Over the past few years, many organizations have designed and developed service discovery protocols. Examples in academia include the Massachusetts Institute of Technology's Intentional Naming System (INS),[1] University of California at Berkeley's Ninja Service Discovery Service (SDS),[2] and IBM Research's DEAPspace.[3] Major software vendors ship their service discovery protocols with their current operating systems—for example, Sun Microsystems' Jini Network Technology,[4] Microsoft's Universal Plug and Play (UPnP),[5] and Apple's Rendezvous.[6] Other organizations have also proposed discovery protocols standards, including Salutation Consortium's Salutation protocol,[7] Internet Engineering Task Force's Service Location Protocol (SLP),[8] and Bluetooth Special Interest Group's Bluetooth SDP.[9]

All these protocols support service discovery in ambient computing environments in terms of network topology or location. Each one addresses a different mix of issues, but most are designed for home or enterprise environments and thus don't always apply to pervasive computing beyond these confines. However, the various design approaches provide useful reference points for future protocol design. We have therefore developed a taxonomy of existing protocols as a basis for analyzing their approaches and identifying problems and open issues relative to service discovery in pervasive computing environments.

Pervasive environment challenges

Pervasive computing environments are far more dynamic and heterogeneous than enterprise environments. Enterprise network services operate within a network scope protected by firewalls and managed by system administrators. But a network scope can't easily define the ambient ser-

1536-1268/05/$20.00 © 2005 IEEE ■ *Published by the IEEE CS and IEEE ComSoc*

vices in pervasive computing environments, nor are all services likely to be managed by system administrators. Pervasive service discovery in an office environment might target an enterprise computing service, but it might also target a service that doesn't belong to the enterprise domain—for instance, a colleague's personal service.

Ambient services discovery also differs significantly from Web services discovery over the Internet. For example, Web services have no physical location limitations, whereas ambient services are local. Moreover, current Web service architectures focus only on interoperations among applications—not people. For example, the World Wide Web Consortium defines its Web services architecture (www.w3.org/TR/ws-arch) to facilitate platform interoperability through standards such as the Web Service Description Language (WSDL)[10] and XML. The discovery and configuration process for Universal Description, Discovery, and Integration[11]—a representative Web services discovery protocol—can involve analysts, programmers, and administrators. This limits UDDI's applicability in pervasive computing environments. Furthermore, its registry and data structure focus on electronic commerce, which is too specific for pervasive computing services.

Pervasive computing environments pose unique questions relative to integration with people and their ambient environments.

Integration with people

Perhaps the most serious challenge to pervasive computing service discovery is the integration of computing devices with people. For one thing, how do we protect personal privacy? When computing devices are integrated with people, various personal information is expressed in digital form. Service discovery protocols can communicate this information over networks—either directly or by inference. For example, you can infer a person's presence information from the RFID tags on clothes or the devices carried in a hand bag. Similarly, you can infer a person's health status from discovering a wearable medical device. Moreover, users expose their intentions in service discovery—to untargeted as well as targeted services. These possibilities impose complex requirements on system design.

Pervasive computing environments pose unique questions relative to integration with people and their ambient environments.

Another question has to do with how much prior knowledge a user or service provider must have for service discovery. Most existing protocols consider only interactions among computers and programs that are acting as clients (discovering devices), services (network services), and directories (centralized service information storage). Nevertheless, people serve two roles in discovery processes: as users of services via clients and as service providers. One end of the spectrum requires little knowledge—users can acquire information without actively managing pervasive computing devices. The other end requires special skills. For instance, many existing service discovery protocols standardize service names and attributes to eliminate ambiguity when, say, a client looks for a "print" service while a service provides a "printing" service. Does this mean that a user must learn service terminology?

Finally, pervasive computing environments allow multiple service providers to coexist at a place. What user roles and administrative domains are properly involved in a discovery? For example, in Bob's office, he might own services and so might his colleague Alice, the office, or the landlord. The credentials he uses to access Alice's MP3 player differ from those he uses to access the office printer. You can't assume that users will provide proper credentials as they do in traditional computing environments because supplying them requires knowledge of the relations among services, service providers, and the user's credentials. The relations can be complex or ultimately unmanageable when a user interacts with thousands of dynamic services that are owned by dozens of service providers.

Integration with environments

Integrating computing devices with environments raises an interesting question: how can we precisely define the ambient environment that a service discovery instance targets?

The primary service discovery protocol approaches are based on LANs and single-hop wireless communication.[4–5,9] These approaches are simple and effective for, respectively, enterprise environments that are behind firewalls and devices that communicate via ad hoc networks. However, they can be too coarse for pervasive computing environments, which are more appropriately defined by physical-world boundaries,[12] such as a room, and other high-level context information, such as location and current user tasks. Moreover, different users in the same place can have different views of the augmented environments. For instance, a visitor's view differs from a host's view.

In pervasive computing, heterogeneity occurs in many aspects: hardware, software platforms, network protocols, and service providers. Future devices and services will increase the diversity. Although

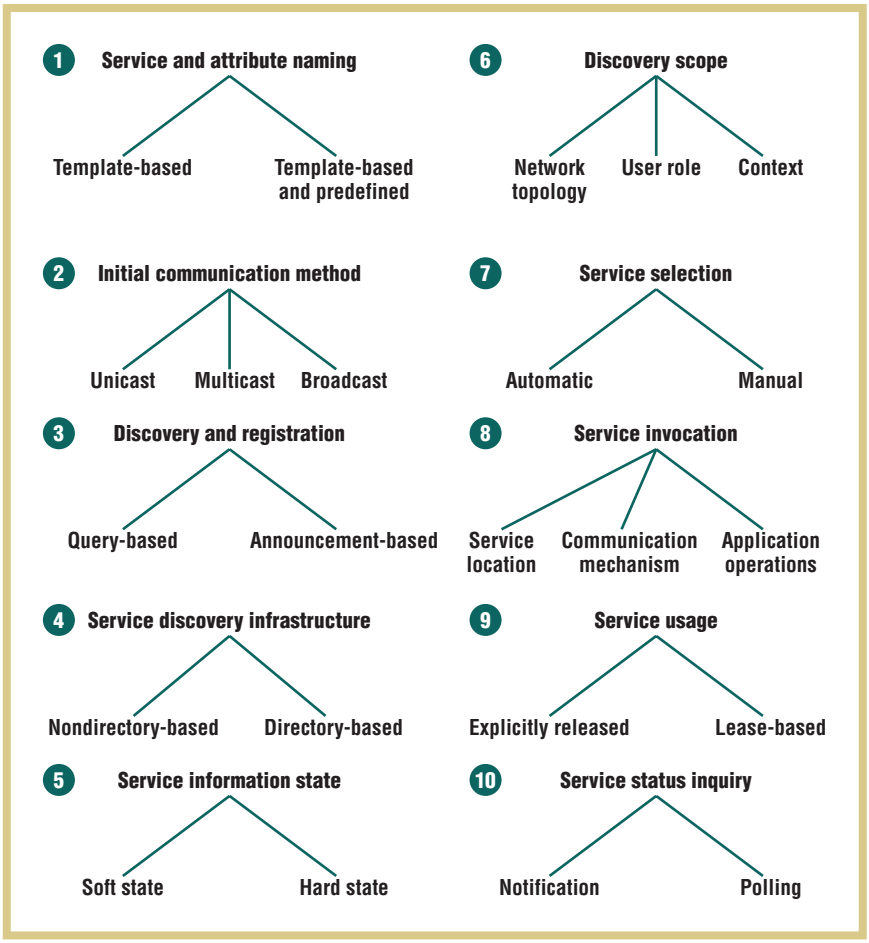


Figure 1. Major service discovery protocol components.

service discovery protocols accommodate heterogeneity, existing protocols have various design goals and solutions. Each has its advantages and disadvantages in different situations, so it seems unlikely that a single protocol could dominate in pervasive computing environments. With current protocols, this means clients and services can't discover each other if they don't use a common protocol. We should therefore establish a common platform to enable interoperability among service discovery protocols.

Are the dynamic conditions typical in pervasive computing qualitatively different from those in mobile computing systems? They seem at least to reach an extreme in pervasive computing: unattended tiny devices, user and device mobility, changing user roles and tasks, and partial failure on devices and applications. Service discovery results can be inconsistent in consecutive queries, and service states can be uncertain. When inconsistency and uncertainty rates are high, users may perceive the system as unstable. Many service discovery protocols adopt time-based approaches (soft states and lease-based approaches) to address dynamic conditions. These approaches simplify distributed system designs to handle dynamic conditions. However, quantitative design analysis might be difficult without actual pervasive computing environments to measure.

Service discovery designs

Existing service discovery protocols often use different terminologies to better reflect unique aspects of their design choices. A common terminology and classification mechanism help analyze their advantages and disadvantages and identify areas worthy of additional efforts to meet pervasive computing environment requirements.

Figure 1 shows major service discovery protocol components and classifies the design choices for them in a consistent taxonomy. Table 1 compares nine protocols relative to this scheme. The table also addresses security and privacy issues.

Service and attribute naming

In the discovery process, a client searches a service by specifying a service name and attributes. This approach has much lower administrative overhead than traditional network service access, which requires prior knowledge of a service's existence and manual inputs such as computer names. Because pervasive computing environments might include hundreds of devices and services, manually configuring them for potential service accesses seems infeasible.

To provide expressiveness for various services and expandability for new services, many discovery protocols use a *template-based* approach to naming—that is, they define a format for service names and attributes. For instance, Apple's Rendezvous (based on the Internet's Domain Name System) defines how a service name must be composed.

In addition to template-based naming, several protocols offer a *predefined* set of common attributes and frequently used service names. For example, Bluetooth SDP defines a set of 128-bit Universal Unique Identifiers for frequently used services and a set of attribute IDs. This eliminates ambiguity regarding names and attributes when clients, services, and directories interact.

Several protocols, such as Jini and Bluetooth SDP, offer user-friendly service names and attributes besides the machine-friendly versions. Nevertheless, users still need to learn the terms to use services.

The various templates and predefined service names and attributes mean that pervasive computing environments face the problem of how clients, services, and

TABLE 1
Comparison of service discovery protocols according to major component categories and security issues.

	Research community			Software vendors
Feature	**INS**	**Ninja SDS**	**DEAPspace**	**Jini**
Service and attribute naming	Not available	Not available	Not available	Template-based and predefined
Initial communication method	Unicast and multicast	Unicast, multicast, and broadcast	Broadcast	Unicast and multicast
Discovery and registration	Query- and announcement-based	Query- and announcement-based	Announcement-based	Query- and announcement-based
Service discovery infrastructure	Directory-based, flat or hierarchical*	Directory-based, hierarchical	Nondirectory-based	Directory-based, flat or hierarchical
Service information state	Soft state	Soft state and hard state‡	Soft state	Soft state
Discovery scope	User role (administrative domain)	User role (administrative domain), context (location), and network topology (LAN)	Network topology (single-hop ad hoc network)	User role (administrative domain), context (location), and network topology (LAN)
Service selection	Automatic	Manual	Manual	Manual
Service invocation	Not available	Not available	Not available	Service location and communication mechanism (downloadable Java code and RMI)
Service usage	Not available	Not available	Not available	Lease-based
Service status inquiry	Not available	Not available	Not available	Notification
Security dependency	Not available	Built-in	Not available	Built-in
Authentication	Not available	Yes (user and service)	Not available	Yes (user and service)
Authorization	Not available	Yes (access control list and capability)	Not available	Yes
Confidentiality	Not available	Yes	Not available	No
Integrity	Not available	Yes	Not available	Yes
Availability	Not available	Yes	Not available	No
Nonrepudiation	Not available	Yes	Not available	No
Privacy	Not available	Service privacy	Not available	No

* Flat in subdomain and hierarchical globally.
† Multiple directories in SLP do not communicate with each other. Coexisting directories are designed for fault tolerance and performance.
‡ Soft state at leaf directories; hard state at other directories.

directories can understand each other when more than one discovery protocol is widely adopted.

Initial communication method

Unicast is the most efficient initial communication method among clients, services, and directories because it involves only related parties. Its drawback is a requirement to configure network addresses with prior knowledge. Such configuration might be infeasible to do manually in pervasive computing.

An elegant solution in current service discovery protocols is User Datagram Protocol *multicast*. After clients, services, and directories receive a few UDP multicast messages, they determine unicast addresses dynamically and switch communications to unicast. In this way, they only need to set a few multicast addresses initially. They can also save the unicast addresses for future usage and performance optimization.

An alternative method is link layer *broadcast*. Broadcast has the same advantage as multicast, but it is usually limited within a single hop of wired or wireless networks.

Binding a service discovery protocol tightly to the underlying network protocols can limit its discovery capability. For example, a device with only a Blue-

		Industry standards community		
UPnP	**Rendezvous**	**Salutation**	**SLP**	**Bluetooth SDP**
Template-based and predefined	Template-based	Template-based and predefined	Template-based	Template-based and predefined
Unicast and multicast	Unicast and multicast	Unicast and broadcast	Unicast, multicast, and broadcast	Unicast and broadcast
Query- and announcement-based	Query-based	Query- and announcement-based	Query- and announcement-based	Query-based
Nondirectory-based	Directory-based, hierarchical	Directory-based, flat	Nondirectory- or directory-based†	Nondirectory-based
Soft state	Soft state and hard state	Hard state	Soft state	Soft state
Network topology (LAN)	User role (administrative domain)	Network topology (LAN)	User role and network topology (LAN)	Network topology (single-hop ad-hoc network)
Manual	Manual	Manual	Manual	Manual
Service location, communication mechanism (XML, SOAP, and HTTP), and application operations	Service location (TCP/IP address and port)	Service location, communication mechanism (RPC), and application operations	Service location (URL)	Service location
Explicitly released	Not available	Explicitly released	Explicitly released	Not available
Notification and polling	Not available	Notification	Not available	Not available
UPnP Security	DNS Security Extension	Built-in	Built-in	Built-in and Bluetooth Security
Yes (user, service, and device)	Yes (service information)	Yes (user)	Yes (service information)	Yes (device authentication)
Yes (Access control list and certificates)	No	Yes	No	Yes
Yes	No	No	No	Yes
Yes	Yes	No	Yes	No
No	No	No	No	No
No	No	No	No	No
No	No	No	No	No

tooth network interface won't be discovered by any devices that lack one. Salutation provides an innovative solution by using two interfaces: one that lets applications call service discovery functions and the other that makes the Salutation transport layer independent. A mapping from Salutation to Bluetooth SDP enables discovery of Bluetooth devices.[13]

Discovery and registration

Clients, services, and directories have two basic options for exchanging discovery and registration information. Many protocols support both approaches.

In the *announcement-based* approach, interested parties listen on a channel. When a service announces its availability and information, all parties hear the information. So in this approach, a client might learn that the service exists and a directory might register the service's information.

In the *query-based* approach, a party receives an immediate response to a query and doesn't need to process unrelated announcements. Multiple queries asking for the same information are answered separately.

Service discovery infrastructure

Protocols choose between directory- and nondirectory-based infrastructure models.

The *directory-based* model has a dedicated component: a directory that maintains service information and processes queries and announcements. Some directories provide additional functionality. For instance, Ninja SDS directories support secure announcements and queries.

The *nondirectory-based* model has no dedicated component. When a query arrives, every service processes it. If the service matches the query, it replies. When hearing a service announcement, a client can record service information for future use.

The nondirectory-based model is suitable for simple environments such as individual homes where the services are relatively few. The directory-based model is more suitable for environments with hundreds or thousands of services, where a directory can process client queries more efficiently.

SLP and Salutation provide more flexible solutions. SLP supports both the directory-based and nondirectory-based models. Although the initial discovery stage incurs some overhead to determine the existing discovery model, such flexibility is preferable in heterogeneous pervasive computing environments. Salutation has a component, Salutation Manager, that integrates the service discovery functions of a client, service, and directory into one component. This enables one device to act according to different roles in different situations.

Directory-based models fall into two design categories: flat and hierarchical structures. In a *flat directory structure*, directories have peer-to-peer relationships. For example, within an INS subdomain, directories have a mesh structure: a directory exchanges information with all other directories. On the other hand, Rendezvous and Ninja SDS have a tree-like *hierarchical directory structure*. Because pervasive computing environments often integrate services with ambient environments or people, a directory must determine whether the service information that it exchanges with a neighbor or a parent directory is appropriate in terms of locality, security, and privacy.

Service information state

Most service discovery protocols maintain service status as a *soft state*. A service announcement specifies the service's life span. Before the service expires, a client or directory can poll the service for validity or the service can announce itself again to renew the registration lease. Otherwise, the service will become invalid after expiration. The directory-based model removes expired service entries from the directories. This graceful, soft-state, service management mechanism greatly simplifies system design and keeps service information fresh.

A directory must determine whether service information that it exchanges with other directories is appropriate in terms of locality, security, and privacy.

Alternatively, clients and directories can maintain service status as a *hard state*. Hard state requires few service announcements and housekeeping jobs. However, it does require the clients and directories to periodically poll services to make sure that their information is up-to-date.

Discovery scope

Proper discovery scopes minimize unnecessary computation on clients, services, and directories. Scopes based on network topologies, user roles, context information, or combinations of such information help to properly define service discovery session targets.

In *network topology*-based discovery scopes, some protocols use LANs as the default, while others use a single-hop wireless network range. An implicit assumption is that the clients, services, and directories are in a place that belongs to the same administrative domain. The assumption might hold true in enterprise and home computing environments, but pervasive computing environments might include multiple, coexisting administrative domains as well as different underlying networks that might not be connected.

User roles offer another approach. Many service discovery protocols support administrative domains as a discovery scope. Users either authenticate with a domain or supply the designated domain as an attribute. This lets users control the target domain, but it requires them to have prior knowledge of the target service and its domain. A fine-grained implementation of this approach should reflect an ambient environment according to a user's roles instead of reflecting the same augmented environment to all users.

High-level *context* information such as temporal, spatial, and user activity information can also help define the discovery scope. Proper use of context information will save users time and effort in discovering services. However, few protocols integrate context information in simple forms. For example, a Jini service query can specify a physical location as an attribute. In brief, graceful integration with context-aware systems and methods for utilizing imperfect context information still need much research.

Service selection

Although discovery scopes limit the number of service matches, a discovery result might still contain a list of matched services. When this happens, service discovery protocols can offer manual or automatic selection options. A completely *manual* mode gives users

total control, prompting them to select a list of services. However, the selection process can be tedious and users might not know enough about the services to distinguish among them.

At the other extreme, with *automatic* mode, the service discovery protocols select the service. This approach simplifies client programs or requires little user involvement. In INS, a service lookup is resolved to a service location at the delivery time. When a set of similar services meet a client's request, a service request is routed to one of the services according to an application-defined service weight. INS calls this *anycast*. This approach utilizes service-side knowledge and helps balance the load among services. Nevertheless, the target service might not meet the user's intent because it's challenging to automatically and accurately fit a need that the user hasn't explicitly specified in attributes.

The Condor project[14] has developed a sophisticated service selection mechanism, Matchmaking with ClassAds, that supports attribute matching even if a query hasn't explicitly specified all attributes. The mechanism also allows both discovering parties and service providers to specify their preferences. In short, Matchmaking with ClassAds can express a wide range of resource information without a schema.

Service invocation

After service selection, a client invokes the service. Invocation involves a service's network address, an underlying communication mechanism, and operations specific to an application domain. Existing service discovery protocols provide three different support levels.

At the first level, protocols provide only network addresses—that is, *service location*; applications are responsible for defining the communication mechanism and operations. At the second level, in addition to service location, a protocol defines the underlying *communication mechanisms*—Remote Procedure Call and its variations. For example, Jini uses Remote Method Invocation and downloadable Java code. The Java code moves from a service to a directory and then to a client. Next, the client calls the service via the downloaded code. UPnP's communication mechanism is based on XML, SOAP, and HTTP. The platform-independent UPnP protocol stack increases interoperability. WSDL is based on a similar underlying communication mechanism: XML, SOAP, and HTTP. WSDL also provides a model to describe operations, message flows, fault handling, and plug-in components.

At the third level, Salutation and UPnP define *application operations* specific to an application domain in addition to the communication mechanism and service location.

Pervasive computing environments might require a more general approach to service invocation because they can't guarantee that a client and service have prior application-domain knowledge to understand and interact with each other. Such an approach also establishes a general interface for introducing new devices and services. Pioneering work such as HP Cooltown[15] uses Web pages as the general information format, and clients and services exchange URLs for addressing. It therefore requires minimal prior knowledge. Researchers at the Palo Alto Research Center proposed recombinant computing,[16] a concept that provides a set of general interfaces for arbitrary devices or services to interact. In the PARC implementation, the interfaces allow services and devices to set up connections and control behaviors. Devices and services understand each other by explicitly exchanging their context information. This approach lets users decide what devices or services should interact instead of having application designers make the decisions.

Service usage

In some service discovery protocols, a client must *explicitly release* a service's resources once service usage is granted. Unlike those protocols, Jini uses a *lease-based* mechanism. A client and a service negotiate a service usage period, which the client can later cancel or renew. Thus, the client and service know that resources will be reclaimed when leases expire. Information is deleted after leases expire so that services won't accumulate stale information, which can occur in traditional distributed systems when a client crashes. Therefore, lease-based service usage handles dynamic conditions better in pervasive computing environments.

A general approach to service invocation in pervasive computing environments also establishes a general interface for introducing new devices and services.

Service status inquiry

A client can keep up with a service's events or status by *polling* it periodically. Another way is through *service event notification*. In this method, clients register with a service, and the service then notifies them when something of interest occurs. If a service generates events frequently or changes status quickly, it's better to use service polling.

It's even better to have agents filter and aggregate events. Jini provides several

ways to do this. A service can send events to agents and let them ensure delivery to clients, or an agent can act as a sink for events that it filters, aggregates, and then sends to clients. An agent can also resemble a mailbox that filters events over time. Although event filtering and aggregation benefit both clients and services, they require some network computer resources to handle the events.

spective, service information, identities, and presence information might be sensitive. From a user's perspective, so are authentication credentials and service query information.

While keeping in mind that environments change, we revisit the service discovery components.

In discovery and registration, clients, services, and directories should exchange sensitive information only with legitimate parties in the vicinity. Legitimacy refers to both valid credentials and access privileges on services. For example, Bob's office can be a legitimate service provider to him in terms of printing services but not in terms of pop songs. Unfortunately, legitimacy might not be easy to acquire. Users that are discovering services might not know of existing services and service providers prior to discovery. Similarly, pervasive computing's mobility and dynamic characteristics might keep service providers from having prior knowledge of legitimate users. We've expressed the legitimacy problem as a chicken-and-egg problem elsewhere[19] and proposed a progressive-exposure approach where users and service providers progressively exchange credential and service information to determine legitimacy.

Rethinking service discovery infrastructures from users' and service providers' perspectives reveals how much improvement is necessary for security, especially in proper interaction between humans and computing devices. Current designs focus on interactions among clients, services, and directories. From a user's perspective, however, a discovery should reflect a user's roles instead of a client device. For example, Bob and his son should discover different services at home when using the same PDA. More challenging, a user might want a client device to interact with many services, each requiring different kinds of credentials.

Moreover, everyone can own many services and become a service provider when computing devices are integrated with their belongings. From a service provider's perspective, managing and sharing services via a service discovery protocol requires system software that provides simple user interfaces and powerful tools to manage security and privacy. Carl Ellison found that a security policy for a home environment could be much more complex than for an enterprise environment.[20] Building a tool that lets users learn who has accessed or is accessing a service is also desirable.

Because clients can invoke a service over insecure networks, we should use end-to-end encryption to protect client-service interactions. Additionally, prudent interface design and implementation for clients and services are necessary because predicting all potential interactions between a client and service is difficult.

In comparison to service discovery functionality, support for security and privacy in existing service discovery protocols is still at an early stage. In table 1, we compare the protocols on authentication, authorization, confidentiality, integrity, availability, nonrepudiation, and privacy. Most secure protocols shown in the table directly use existing security solutions, which means that many pervasive computing requirements aren't met. However, recent revisions of the service discovery protocols and new protocols are supporting more security features. The "Secure Service Discovery Protocols" sidebar describes representative secure service discovery protocols.

Rethinking service discovery infrastructures from users' and service providers' perspectives reveals how much improvement is necessary for security.

Heterogeneous device support

Many devices in pervasive computing environments have limited resources and so don't have service discovery capabilities. Sun Microsystems designed Jini Surrogate[17] to assist those devices. A Jini-capable machine hosts a Java object (surrogate) to represent a device for Jini service discovery.

Security and privacy

Service discovery protocols need security and privacy features to protect devices, services, and users. Nevertheless, existing security solutions might not directly apply. Ross Anderson pointed out that many security solutions fail when applied to different environments, simply because the environments change.[18] Furthermore, many current solutions require people to have special skills.

Unlike an enterprise environment, where physical boundaries and firewalls can separate outsiders and insiders, users and service providers can coexist in a pervasive computing environment and interact via wireless networks. Furthermore, personal privacy remains a critical issue. From a service provider's per-

Secure Service Discovery Protocols

Bluetooth Security[1] defines a profile for service discovery applications that augments the security Bluetooth provides at the link layer. Service discovery is classified as trusted and untrusted. In *trusted discovery*, service information is exposed only to a device that shares a common secret with the service. In *untrusted discovery*, service information is available to any device. In pervasive computing, requiring two parties to share a secret beforehand might be inconvenient or even infeasible. Thus, a high-level protocol to distribute secrets to Bluetooth devices might be desirable.

Unlike other secure service discovery protocols, UPnP Security defines interactions among not only computers but also people and environments.[2] For example, it specifies a security procedure for a user to take ownership of a device. By standardizing the human-computer interaction procedure, UPnP Security reduces many security problems caused by human operations. Moreover, because it is designed for heterogeneous devices and services, UPnP Security supports many authorization methods: access control lists, authorization servers, authorization certificates, and group definition certificates. In short, UPnP Security addresses many security issues for pervasive computing service discovery.

Jini provides an abstract, extensible interface by which application designers or system administrators can use various security protocols to support other security requirements such as authentication and confidentiality. In addition, Jini addresses new security issues raised in its service invocation—specifically, verifying the trustworthiness and granting permissions to downloadable Java code.[3]

Ninja SDS is a centralized security solution. It supports authentication, authorization, data and service privacy, and integrity. Directories, known as SDS servers, are trusted. Clients and services authenticate with the directories for service lookups and announcements, respectively. Ninja SDS is good for enterprise environments where users and services are willing to expose information to central directories. In pervasive computing, however, trusted central directories pose personal privacy problems because every service registers with central directories and every request is sent to central directories.

PrudentExposure addresses security and privacy issues when multiple service providers coexist.[4] The discovery is user-centric—that is, based on user roles. By exchanging a little information, an agent can select pertinent credentials to authenticate with service providers automatically. The design simplifies the security requirements on clients and frees users from memorizing the relationships among services, service providers, and their own credentials. PrudentExposure also protects sensitive information by not responding to arbitrary requests unless some trust information is exchanged.

REFERENCES

1. Bluetooth SIG Security Expert Group, "Bluetooth Security White Paper," Apr. 2002; http://grouper.ieee.org/groups/1451/5/Comparison%20of%20PHY/Bluetooth_24Security_Paper.pdf.
2. C. Ellison, "UPnP Security Ceremonies V1.0," Oct. 2003; www.upnp.org/download/standardizeddcps/UPnPSecurityCeremonies_1_0secure.pdf.
3. F. Sommers, "Jini Starter Kit 2.0 Tightens Jini's Security Framework," *Java World*, May 2003; www.javaworld.com/javaworld/jw-05-2003/jw-0509-jiniology.html.
4. F. Zhu, M. Mutka, and L. Ni, "A Private, Secure and User-Centric Information Exposure Model for Service Discovery Protocols," to be published in *IEEE Trans. Mobile Computing*.

Obviously, many problems need further research. Among the problems, service discovery for unfamiliar computing environments hasn't been addressed well. However, service discovery protocols must work in unfamiliar computing environments to achieve the goal of computing anytime and anywhere. Service discovery design for such environments is more challenging: users don't have much prior knowledge, most don't have special skills, and they might not trust the environment. This means that service discovery protocols and the underlying computing infrastructure must have more intelligence.

ACKNOWLEDGMENTS

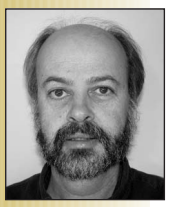
We thank the anonymous editor and reviewers for their valuable comments to help us improve this article. The research was supported in part by US National Science Foundation grant 0334035, and Hong Kong Research Grants Council grants HKUST6161/03E and AoE/E-01/99.

REFERENCES

1. W. Adjie-Winoto et al., "The Design and Implementation of an Intentional Naming System," *Proc. 17th ACM Symp. Operating System Principles* (SOSP 99), ACM Press, 1999, pp. 186–201.
2. T. Hodes et al., "An Architecture for Secure Wide-Area Service Discovery," *ACM Wireless Networks J.*, vol. 8, nos. 2/3, 2002, pp. 213–230.
3. M. Nidd, "Service Discovery in DEAPspace," *IEEE Personal Comm.*, Aug. 2001, pp. 39–45.
4. *Jini Technology Core Platform Specification*, v. 2.0, Sun Microsystems, June 2003; www.sun.com/software/jini/specs/core2_0.pdf.
5. *UPnP Device Architecture 1.0*, UPnP Forum, Dec. 2003; www.upnp.org/resources/documents/CleanUPnPDA10120031202s.pdf.
6. S. Cheshire and M. Krochmal, "DNS-Based Service Discovery," IETF Internet draft, work in progress, June 2005.
7. *Salutation Architecture Specification*, Salutation Consortium, 1999.
8. E. Guttman et al., *Service Location Protocol*, v. 2, IETF RFC 2608, June 1999; www.ietf.org/rfc/rfc2608.txt.

the AUTHORS

Feng Zhu is a PhD candidate in computer science and engineering at Michigan State University. His research interests include pervasive computing, security for pervasive computing, computer networks, and distributed systems. He received his MS in computer science and engineering from Michigan State University. He is an IEEE student member. Contact him at the Dept. of Computer Science and Eng., 3115 Eng. Bldg., Michigan State Univ., East Lansing, MI 48824; zhufeng@cse.msu.edu.



Matt W. Mutka is an associate professor in the Department of Computer Science and Engineering at Michigan State University. His research interests include mobile computing, wireless networking, and multimedia networking. He received his PhD in computer science from the University of Wisconsin-Madison. He is an IEEE senior member and a member of the ACM. Contact him at the Dept. of Computer Science and Eng., 3115 Eng. Bldg., Michigan State Univ., East Lansing, MI 48824; mutka@cse.msu.edu.

Lionel M. Ni is chair professor and head of the Computer Science Department at the Hong Kong University of Science and Technology. His research interests include wireless sensor networks, distributed systems, high-speed networks, and pervasive computing. He earned his PhD in electrical and computer engineering from Purdue University. He is an IEEE fellow and a member of the ACM and SIAM. Contact him at Dept. of Computer Science, Hong Kong Univ. of Science and Technology, Clear Water Bay, Kowloon, Hong Kong; ni@cs.ust.hk.

9. *Specification of the Bluetooth System*, Bluetooth SIG, Feb. 2003.
10. R. Chinnici et al., *Web Services Description Language (WSDL) Version 2.0*," W3C working draft, Aug. 2004; www.w3.org/TR/2004/WD-wsdl20-20040803.
11. *UDDI Version 2.04 API Specification*, OASIS standard, July 2002; http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.pdf.
12. T. Kindberg and A. Fox, "System Software for Ubiquitous Computing," *IEEE Pervasive Computing*, vol. 1, no. 1, 2002, pp. 70–81.
13. B. Miller, "Mapping Salutation Architecture APIs to Bluetooth Service Discovery Layer, Version 1.0," white paper, 1999; http://citeseer.ist.psu.edu/251752.html.
14. M. Litzkow, M. Livny, and M.W. Mutka, "Condor—A Hunter of Idle Workstations," *Proc. 8th IEEE Int'l Conf. Distributed Computing Systems*, IEEE Press, June 1988, pp. 104–111.
15. T. Kindberg et al., "People, Places, Things: Web Presence for the Real World," *Proc. 3rd IEEE Workshop Mobile Computing Systems and Applications* (WMCSA 00), IEEE CS Press, 2000, p. 19–28.
16. K. Edwards, M. Newman, and J. Sedivy, *The Case for Recombinant Computing*, tech. report CSL-01-1, Palo Alto Research Center, Apr. 2001.
17. *Jini Technology Surrogate Architecture Specification*, Sun Microsystems, July 2001; http://surrogate.jini.org/sa.pdf.
18. R. Anderson, *Security Engineering: A Guide to Building Dependable Distributed Systems*, John Wiley & Sons, 2001.
19. F. Zhu et al., "Expose or Not? A Progressive Exposure Approach for Service Discovery in Pervasive Computing Environments," *Proc. 3rd IEEE Int'l Conf. Pervasive Computing and Communications* (PerCom 2005), IEEE CS Press, 2005, pp. 225–234.
20. C. Ellison, "Home Network Security," *Intel Tech. J.*, vol. 6, no. 4, 2002, pp. 37–48.