Discovery Systems in Ubiquitous Computing

Ubiquitous computing introduces unique requirements for discovery technologies, which let services and devices become aware of each other without explicit human administration.

W. Keith Edwards
Georgia Institute of Technology

The vision of ubiquitous computing has been described in terms of the disappearing computer. In this vision, we're free to focus on the interactions of daily life, rather than attending to the technology assisting us in those interactions. The machine—while perhaps not literally invisible—becomes a tool much like a hammer or pencil, easily appropriated and used as second nature.

To achieve this effective invisibility, our systems should be as free as possible from explicit human administration. This is especially challenging in ubiquitous computing, where devices can dynamically enter and leave the network. If all the devices require explicit configuration to work together, the burden of administration quickly overwhelms any potential benefit.

Discovery lets services and devices spontaneously become aware of the availability and capability of peers on the network without explicit administration. In practice, this means that a client can discover and potentially use a device without prior knowledge of it. Although discovery is a necessary component of ubiquitous computing, the wide range of discovery systems in use today reflects the varied needs of the communities from which they originated. Some of these features are appropriate for ubiquitous computing, but others require further research and development.

What is discovery?

Discovery is the process by which an entity on a network (a *client*) is spontaneously notified of the availability of desirable services or devices on the network (*resources*). More precisely, discovery is a mechanism for dynamically referencing a resource on the network. These references are handles or other information that the client can subsequently use to contact the resource.

Resources entering the network make themselves available by registering with the discovery system. This can involve finding a directory service and registering with it or simply making periodic announcements on the network. During this process, resources provide descriptive information (such as their resource type and attributes) as well as information the client will need to use them (such as an IP address and port number). Clients provide criteria describing the resources they're interested in, and the system uses these criteria to identify appropriate resources for the client. This process might involve querying a directory or simply filtering resource announcements. In all discovery systems, registration and deregistration are highly dynamic—as resources come and go, the discovery system can asynchronously notify clients of resources' availability.

This spontaneity makes discovery systems vital in a ubiquitous computing setting.[1] Clients find resources automatically, rather than needing preconfigured bindings to specific resources. Spontaneity also helps differentiate discovery systems from naming systems, which on the surface are similar in that

- both systems let clients retrieve references to resources,
- both typically provide mechanisms for describing resources, and
- many of these systems let clients search on the basis of descriptive criteria.

Published by the IEEE CS and IEEE ComSoc ■ 1536-1268/06/$20.00 © 2006 IEEE

Often, architectural similarities also exist. For example, naming systems use a potentially distributed directory on the network that serves as a registry of information about available resources. Many discovery systems also use directories.

True discovery systems differ from naming systems in three major ways:

- In most naming systems, the directory's location must be preconfigured into clients and resources. In discovery systems, clients and resources automatically find the directory—if one is present at all.
- In naming systems, updating the directory typically requires explicit human intervention. As resources come and go, an administrator must edit tables to reflect the network's new configuration. This process is automatic in discovery systems.
- Few naming systems have a way to notify clients about changes in the set of registered resources, so clients must poll the directory. Discovery systems can asynchronously notify clients as resources come and go.

On the basis of these criteria, systems such as the Lightweight Directory Access Protocol,[2] the Domain Name System,[3] and the Universal Description, Discovery, and Integration (UDDI)[4] protocol (despite its name) clearly aren't true discovery systems. For example, LDAP provides no automatic mechanism for adding or removing resources in the directory. Likewise, although standard DNS provides a powerful, hierarchical name service, it lacks certain features required to support discovery. New protocols can and have supplied some of these features. For example, the Dynamic Host Configuration Protocol (DHCP) mitigates the requirement that DNS server addresses must be preconfigured into clients, and Dynamic DNS lets hosts automatically update their DNS registrations. Still, traditional DNS lacks the automatic update and asynchronous notifications needed for discovery (as we shall see, however, recent extensions to DNS make it suitable as a discovery protocol).

Discovery-based systems present a more dynamic, fluid interaction model than naming-based systems. Naming-based systems generally expect applications to know the name of the thing for which they wish to retrieve a reference. Discovery-based systems dynamically provide references, at roughly the time the resource becomes available.

Characterizing discovery systems

Although discovery systems' high-level goals are largely the same, individual systems achieve these goals differently. Variations arise from differences in application domain and usage scenarios. The most important dimensions characterizing discovery systems are topology, transport, scope, search, and security.

Topology

Different technologies rely on different topologies. Some systems use one or more directories to handle resource registration and client queries. Although these systems require extra administration to manage the directory, they generally achieve higher scalability because most traffic is in the form of unicast messages to and from the directory. At the other end of the spectrum are pure peer-to-peer systems. In these systems, clients discover resources directly, rather than indirectly through an intermediating directory. Peer-to-peer systems don't need explicit administration to run the directory, but they can produce higher network loads and require clients to process discovery traffic. Hybrid solutions try to balance these trade-offs for certain expected usage scenarios.

Transport

Some discovery systems essentially operate as middleware on top of the traditional IP infrastructure, letting clients discover and use IP-based resources. Others, such as Bluetooth, are intended for non-IP networks and have different constraints. Still others use out-of-network mechanisms, in which discovery messages are carried over a transport not intended for general data communication. This family of systems includes RFID and infrared (IR) beacons. The set of devices the system aims to support, as well as the devices' power, computation, and connectivity characteristics, often guide the transport choice.

Scope

Scope is simply the set of resources that are discoverable by a given client. In IP-based systems without directories, for instance, scope is generally limited to those resources reachable by an administratively scoped multicast—that is, resources on the client's subnet. Directory-based systems can expand scope by interlinking directories to provide whole site (or larger) scopes. Approaches based on technologies with limited transmission range (RF and IR, for example) inherently limit scope to physical proximity.

Search

Some discovery systems let clients search only on the basis of resource type, whereas others support queries using full-blown predicate languages. This dimension trades computation for flexibility—powerful search mechanisms require substantial computation (in either a directory or in clients themselves) but can allow fine-grained specification of desired resources. Settings with numerous resources or resource types can require such specialized search facilities.

Security

Security mechanisms for discovery systems range from nonexistent to full-blown systems for link-layer encryption,

authentication, and access control. Security always has a cost—of computation or convenience—but can be necessary where trusted and untrusted devices come into contact with each other.

Discovery in practice

The choice of discovery technology depends on the intended application. Many discovery systems were designed with the enterprise in mind and embody design decisions reflecting that intent: managed directories intended to run on centralized servers, IP-based transports compatible with enterprise services, flexible scoping, powerful search mechanisms that execute in the directory service, and security to prevent unwanted access to enterprise resources. While many ubiquitous computing systems will fit into these design choices, others won't. In particular, mobile systems or systems that don't support constant connectivity, that prioritize low power consumption, or that have limited computational capacity might require technology with a different set of trade-offs.

This article's space limitations preclude an exhaustive cataloging of all commercial and research discovery systems, so here I explore a handful of common systems that exemplify distinct design points.

Simple Service Discovery Protocol

Universal Plug and Play,[5] a technology defined by a consortium of vendors including Microsoft, Intel, Sony, and Samsung, uses the Simple Service Discovery Protocol. UPnP leverages common Web technologies, such as HTTP, SOAP, and XML, to provide access to devices and services.

SSDP reflects UPnP's Web-centric focus: it's defined on top of HTTP and identifies resources (and resource types) using uniform resource identifiers. In UPnP's multilayered architecture, devices can host one or more services. For example, UPnP can represent a multifunction printer as a device, and the print, copy, and fax functionalities as individual services. Each service is a discrete, separately callable functional unit. Both devices and services have associated types, expressed as URIs (for example, an Internet gateway device type might be specified as urn:schemas-upnp-org:device:InternetGatewayDevice), and unique IDs.

Jini services and clients can communicate without shared knowledge of a specific protocol through mobile code delivered from a service to a client.

SSDP doesn't use a directory. Instead, it exploits link-local multicast to let clients discover resources directly. So, devices send presence announcements as HTTPMU (HTTP over multicast User Datagram Protocol) messages when they come online. Any party on the local link that's listening on the predefined multicast address will receive these messages. Clients send discovery requests, also over HTTPMU. Devices respond to requests by delivering a reply to the IP address and port number originating the request.

SSDP supports a simple form of searching, in which clients specify the desired resource type's URI. Several special URIs serve as tokens for matching all devices, specific devices by ID, and so forth. SSDP doesn't support searching for multiple types in the same request, nor does it provide attribute-based search. While perhaps somewhat restrictive, this approach makes it extremely economical to implement on simple devices.

Initial presence announcements, as well as responses to discovery requests, carry essentially the same payload: a URI identifying the resource's type, its ID, and a URL pointing to a description document. That is, an XML document providing information about the device and the services it offers, including icons and descriptive text—and URLs for communicating with the device.

Developers have recently extended UPnP with a set of security enhancements, including a secure discovery mechanism that supports authentication of discovered devices and a framework for device access control.

Jini's discovery protocols

Sun Microsystem's Jini network technology is an infrastructure for creating services that adapt to change.[6] For example, Jini services and clients can communicate without shared knowledge of a specific protocol through mobile code delivered from a service to a client. Much of Jini's power comes through using a common executable format (Java) more-or-less ubiquitously throughout the network.

Jini uses a hybrid, or two-stage, topology for discovery. In the simplest case, clients and services use a multicast protocol to find lookup services on their networks. This initial discovery protocol only bootstraps communication with the lookup service, which then defines a set of operations that allows registration, search, and notification about changes in the set of available services.

In Jini, services communicate with clients through *proxy* objects—complete Java objects that the service serializes and transmits over the wire to the client. The proxy lets the service control both communication end points, thereby hiding the actual communication protocol from clients. Because lookup services are also normal Jini services (that is, they also pro-

vide custom code to clients), the initial bootstrapping discovery phase yields a proxy for the lookup service itself. This approach lets different lookup service implementations exist, using different mechanisms to communicate with clients.

Jini's search mechanisms also reflect Java's ubiquitous use. Services register with a lookup service by providing their proxy, as well as attributes that are also full-blown Java objects. Clients can search for services by ID, proxy or attribute Java language types, or attribute contents. The overall result is a solution that, while Java-centric, exploits the assumptions we can make when we have a ubiquitous object description and code execution format. Because this approach can limit the network's heterogeneity, a Java-capable device can act as a surrogate for non-Java devices.[7]

Jini's two-stage discovery mechanism lets clients and services participate in discovery without explicit configuration. It also lets us construct higher-level discovery topologies: because lookup services can register with other lookup services on remote networks (albeit with a bit of configuration), clients can indirectly discover lookup services that would be inaccessible to them via multicast alone. Of course, lookup services must be on the network for discovery to work.

As of version 2, Jini provides extensive service security mechanisms. Because lookup services use the same mechanisms as other Jini services, they can exploit Jini's security framework, which allows authentication, communication encryption, and customizable security policies.

Bluetooth

Bluetooth is a short-range RF-based communication technology.[8] Its relatively low bandwidth (723 Kbits to 2.1 Mbits per second) makes it appropriate for devices such as PDAs, mobile telephones, and mice. Bluetooth devices implement one or more standardized profiles (such as the headset profile for mobile telephony and the human interface device profile for mice and keyboards), which define the operations available to clients.

Of the systems described in this article, only Bluetooth uses a non-IP-based discovery protocol. This is because short-range radio has unique needs. In particular, the need for maximum communication efficiency dictates certain optimizations that an IP-based system might not need. Also, because Bluetooth communication is peer-to-peer, it doesn't assume a fixed network infrastructure. Thus, discoverability is based on actual physical proximity, rather than closeness in the IP routing infrastructure.

Bluetooth communication is peer-to-peer, so it doesn't assume a fixed network infrastructure. Thus, discoverability is based on actual physical proximity, rather than closeness in the IP routing infrastructure.

Bluetooth discovery operates at two distinct layers of the network stack. At the higher layer, the Bluetooth Service Discovery Protocol defines a set of primitives for learning about the services available on a given device. Clients find SDP-capable devices by using the lower-level Link Manager Protocol and Logical Link Control and Adaptation Protocol. In effect, Bluetooth separates device discovery from the discovery of services on a given device.

To initiate Bluetooth discovery, a client sends a low-level inquiry message. Bluetooth devices in the discoverable state respond to this message, sending the client their unique device address. (Bluetooth devices can be in a nondiscoverable state for power management and security reasons.) After a client gathers these addresses, it can separately retrieve human-readable device names from the identified devices. A client need not establish a full communications link just to get a device's name.

After establishing a communication link with a device, a client interacts with the device using SDP to determine its functionality. SDP is defined as a separate profile that lets clients peruse a device's available services. A list of attributes describes each service. Each attribute is a key/value pair describing some aspect of the service—such as unique ID, type, icon, and name.

SDP attributes differ from those of other discovery systems covered in this article, largely because of SDP's efforts to conserve communications bandwidth. Instead of arbitrary strings to denote attribute keys, SDP uses 16-bit IDs. Service type specifications define which IDs are valid as keys, the type of data contained in values, and the semantics of values. Attribute values can be any of several types, including strings, integers, URLs, and sequences of data elements. An attribute value type of particular interest is the 128-bit universally unique identifier (UUID). A service specification defines certain attributes as having values of type UUID. To search for services, clients send a list of UUIDs to a device. Any service on that device that contains every UUID in the search query, regardless of their order or which attribute contains the UUID, matches the query.

So, SDP's search mechanisms use only attributes that the service specification has adopted and reduced to a set of UUIDs. Again, Bluetooth's overall goals—storage, computation, and band-

width efficiency—dictate this approach. A fully arbitrary attribute-search mechanism would require clients to transmit and compare potentially much longer operands and define and implement comparison operators for a wider range of data types. UUID-based search is, in contrast, much more efficient, albeit somewhat limited.

SDP provides no security directly. In Bluetooth, security occurs below the SDP level, encrypting communication at the link layer and restricting communication to peers that share some secret. It most commonly achieves this through a *pairing* procedure, in which a user gives two devices some a priori trust information (typically a personal identification number). After this explicit trust-configuration step, paired devices can both discover one another and exploit link-layer encryption.

Service Location Protocol

The Internet Engineering Task Force's Service Location Protocol focuses on discovery in large managed networks, such as enterprises.[9] As such, it combines several sophisticated techniques useful in such networks, including optional directory services and fine-grained search.

Architecturally, SLP supports two distinct modes that let it span a range of administration requirements and network scales. SLP clients and services can use multicast to directly discover services on their local network. However, SLP also supports the use of *directory agents* as intermediary service registries. If a DA is on a network, clients and services will discover it (through either multicast discovery or explicit configuration) and direct all their registration and query traffic to it. So, SLP will use a directory if one is present but will default to basic multicast discovery if one isn't. The protocol's format is the same in either case, although it sends messages via unicast when communicating with a DA. In larger networks, a DA can serve as a cache for service registrations and a centralization point for query processing.

> Allowing arbitrary hosts to resolve names might be contentious from a security perspective: hosts could potentially respond to requests with bogus information.

SLP services are named via URLs that use the service protocol scheme. For example, a printer might have the name `service:printer://hostname`. SLP supports the use of abstract and concrete service types, allowing a service URL to specify both . For example, a `printer` abstract type might be specialized by an `lpr` concrete type. Such a printer would be represented in an SLP service URL as `service:printer:lpr://hostname`.

Services publish their existence through service registration messages containing their service types, service URL, and one or more attributes. Attributes are simple key/value pairs. Strings can be keys and values can be integers, strings, Booleans, or opaque data blocks.

Clients issue service request messages to find desired services. Clients can search on the basis of the service's type (such as `service:printer`) or attributes. SLP has an exceptionally rich search mechanism: clients provide full LDAP version 3 search filters as predicates to match service attributes.[10] This search language provides Boolean and arithmetic-comparison operators, wild cards, and grouping. These search capabilities reflect SLP's orientation toward enterprise service discovery and relatively heavyweight directories.

In response to clients' service requests, they receive a reply containing the URLs of matching services, time-to-live information to control how they cache discovery results, and—optionally—information to authenticate the service. SLP defines a range of separate (and optional) protocol messages for retrieving service attributes and browsing for all services on a network.

Bonjour

Although popularized by Apple, the Bonjour technology is based on work begun by the IETF's Zeroconf working group (www.zeroconf.org) in 1999. Bonjour's high-level goal is to enable useful networking in the absence of any fixed infrastructure, including such IP stalwarts as DHCP and DNS servers. Bonjour lets two computers, connected only with a crossover Ethernet cable or peer-to-peer wireless network, work together without end users having to twiddle network settings or refer to machines by IP address.

Although the technology goes deeper than just service discovery, discovery is integral to Bonjour. Perhaps the most distinguishing feature of the Bonjour approach to discovery is that it meshes closely with existing Internet standards while operating in the absence of the traditional managed Internet infrastructure.

Bonjour combines multicast DNS (mDNS) with DNS service discovery (DNS-SD) to leverage existing Internet protocols. mDNS piggybacks DNS by defining a new top-level domain: .local. mDNS presumes that names ending in .local. are meaningful only on the local link and are thus analogous to link-local IP addresses (such as 169.254.x.x). mDNS sends any DNS query for a name ending in .local. to a special multicast

TABLE 1
Comparison of current discovery systems.

System	Topology	Transport	Scope	Search	Security
Simple Service Discovery Protocol (SSDP)	Peer-to-peer (P2P)	Unicast HTTP, multicast HTTP	Subnet	Type or ID	Authentication, access control
Jini	Hybrid	Unicast TCP, multicast User Datagram Protocol	Subnet, bridgeable	Type, ID, or attribute	Jini/Java security mechanisms
Bluetooth Service Discovery Protocol (SDP)	P2P	Link Manager Protocol (LMP) and Logical Link Control and Adaptation Protocol (L2CAP)	Proximity (~10 meters)	Type or attribute (universally unique identifier [UUID] only)	Link-level or service-level encryption, authentication
Service Location Protocol (SLP)	P2P or directory	Unicast TCP, multicast UDP	Subnet, bridgeable	Type or attribute (Lightweight Directory Access Protocol v.3 search predicates)	Optional service authentication
Bonjour	P2P	Multicast DNS (mDNS), DNS service discovery (DNS-SD)	Subnet	Type	Optional IP security (IPsec), DNS security extensions (DNSsec)
Salutation	P2P or directory	Open Network Computing remote procedure call (ONC RPC) over arbitrary transports	Depends on transport	Type or attribute	RPC authentication
Intentional Naming Service (INS)	Decentralized, weakly consistent directories	Unicast UDP	Administrative	Attribute domain	None
Ninja Secure Service Discovery Service (SSDS)	Directory	Authenticated remote method invocation (RMI)	Wide area (through hierarchical directories)	XML-based descriptions	Capability-based access control
eSquirt IR	Beacon-and-reader	Infrared	Proximity (~20 centimeters)	None	Physical proximity required
RFID	Beacon-and-reader	Short-range RF	Proximity (~10 cm)	None	Physical proximity required

address. Hosts on the local link that can resolve the name respond with their addresses. The mDNS mechanisms aren't specific to discovery; they merely let hosts resolve names among themselves, rather than through a separate, managed DNS server (which would have to be running, accessible, and configured into clients).

Allowing arbitrary hosts to resolve names might be contentious from a security perspective: hosts could potentially respond to requests with bogus information. So, systems administrators and developers should use existing security technologies such as IP security (IPsec) and DNS security extensions (DNSsec) with mDNS.

DNS-SD is largely a set of naming conventions describing how services will be represented in DNS records. It defines no new operations and no new DNS record types. It's fully compatible with, but doesn't require, mDNS.

Clients use DNS-SD by issuing requests for DNS pointer (PTR) records, indicating the service types they wish to find. DNS-SD names service types using the form **_type._protocol.domain**. So, for example, a client might find all the Web servers in the "example.com" domain by issuing a query for **_http._tcp.example.com**. In this case, _http denotes the application-layer protocol the client wants (services that speak HTTP), _tcp indicates that the service runs over TCP/IP, and example.com denotes the domain to which the query is directed. When DNS-SD is coupled with mDNS, clients can use the .local. domain as well, allowing service discovery to occur without managed DNS services.

The returned PTR records contain *service instance names*—user-friendly names for services matching the query. For example, a query for printers might return "Joan's office printer." In conventional DNS, PTR records typically contain host names; DNS-SD overloads this request so that when the client passes a service type in the query, the query returns service instance names. To resolve a particular service instance name, a client issues a query for DNS service (SVC) records for that name. The rec-

ords returned by the query contain the service's host and port number.

DNS-SD considers the human-readable service instance names to be the services' canonical names. This naming approach presents a subtle difference between DNS-SD and many of the other discovery systems described in this article, which (although they might optionally support human-readable names) use globally unique service IDs to disambiguate services. In other words, these other systems consider the ID to represent the "truth" about service identity. DNS-SD, on the other hand, makes no distinction between the notions that systems and humans use to identify services.

By leveraging existing technology, Bonjour exploits developers' experience with Internet services and protocols. With only a few simple modifications to conventional DNS, Bonjour can provide service discovery on local networks with little or no administration or configuration.

Other systems

Table 1 summarizes how these systems, as well as several others, fit into the larger space of discovery technologies. The others include Salutation, a commercial system used primarily by printer manufacturers,[11] the Massachusetts Institute of Technology's Intentional Naming System,[12] the University of California, Berkeley's Ninja Secure Service Discovery Service,[13] and two out-of-network, proximity-based discovery systems: Cooltown's eSquirt IR-based system[14] and RFID tags.[15] The last two represent physical forms of discovery technology, in which users actively select devices or objects to which they wish to acquire a digital reference (in essence, the user, rather than the technology, provides the search function for these systems).

New research directions

None of the systems I've described in depth were designed specifically to address the needs of ubiquitous computing systems. The following sections outline some of the special requirements of ubiquitous computing; some of these might be achievable by layering on top of existing solutions, while others may require more open-ended research.

Likewise, a discovery technology rooted in ubiquitous computing's needs will likely provide its own search mechanisms based on these needs.

Providing infrastructure, without the infrastructure

Ubiquitous computing systems must be able to support discovery in mobile environments where there might be no fixed network or system infrastructure, and resources might have their homes in different administrative domains, so they have no shared trust or naming information.

From a research perspective, a key challenge is getting a fixed infrastructure's desirable properties—scalability, security, easy administration, shared naming, and so on—without having the infrastructure itself.

Connecting islands of discoverability

Another requirement derives from the complex network topologies typical in ubiquitous computing settings. Using today's discovery technology, a segmented, multitransport network presents multiple isolated islands of discoverability because different networking substrates necessarily impose their own semantics and constraints and inherently define the discovery scope differently.

Such hodge-podge solutions partition resources and force users to understand the scoping constructs used by different protocols (having to understand multicast radius or directory service topology to figure out why a PDA can't see a media server, for example). To provide seamless interconnectivity, we must find ways to bridge the discovery requirements of individual transports in a way that requires little administration and doesn't insist that end users understand low-level networking concepts.

A search appropriate for ubiquitous computing

Another issue for discovery in ubiquitous computing concerns how clients search for resources. Current search mechanisms rely mostly on relatively static attributes that resources publish about themselves. Clients have a range of tools for matching attributes—from predicate languages (SLP) to restricted matching based on prior knowledge of unique tokens (SDP). These search mechanisms are all justified given the intended uses of the various discovery protocols. Likewise, a discovery technology rooted in ubiquitous computing's needs will likely provide its own search mechanisms based on these needs.

For example, how does context-aware computing impact a ubiquitous computing discovery solution? You can easily imagine applications that need to find specific resources on the basis of their users' current status or the interaction history. Do these applications require new features from a discovery system?

The human issue

Discovery is firmly grounded in its networking roots. For example, in many

systems, scoping is determined by multicast radius, and services are identified as URLs. But even though discovery is a technical mechanism, it has implications for users. How does the choice of a discovery system influence the user experience of applications built on top of it? Discovery systems must provide enough information "up the stack" and provide a coherent-enough model so that users can understand the discovery process.

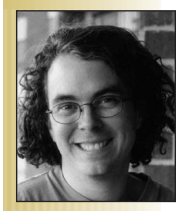
Technical issues such as how to name and find resources have huge ramifications on how users will perceive and work with discovery-based systems. However, surprisingly little work has examined the end-user implications of technical features in discovery. One of the only investigations into such impacts studied the use of music sharing via Bonjour, in which the resource being discovered (a music library) is tightly associated with an individual.[16] In such cases, social issues of privacy and managing the presentation of self come into play. We need to understand how these social effects should influence discovery system design.

Of course, discovery is simply the first step of an application's interaction with some resource. The next, even-more-important step is for the application to use that resource. Most discovery systems simply provide applications with a handle for a resource and then get out of the way. But even if two parties agree on a discovery platform, this in no way guarantees that they'll actually be able to work together. They must further agree on the operations and data formats they will support, as well as the semantics. This isn't a problem for discovery per se, but merely a statement that solving discovery's problems is only a first step. Several research projects have begun to look at more flexible approaches to interoperability. For example, Cooltown leverages Web technologies to allow spontaneous interoperation among devices and services;[14] the Speakeasy project uses a small set of metainterfaces to let devices exchange new behaviors needed for compatibility at runtime;[17] and systems such as Stanford's iRoom use a tuple space as a common, extensible data representation to achieve interoperability.[18]

REFERENCES

1. T. Kindberg and A. Fox, "System Software for Ubiquitous Computing," *IEEE Pervasive Computing*, vol. 1, no. 1, 2002, pp. 70–81.
2. J. Hodges and R. Morgan, *Lightweight Directory Access Protocol (v3)*, IETF RFC 3377, Sept. 2002; www.rfc-editor.org/rfc/rfc3377.txt.
3. P. Mockapetris, *Domain Names—Concepts and Facilities*, IETF RFC 1034, Nov. 1987; www.rfc-editor.org/rfc/rfc1034.txt.
4. *Introduction to UDDI: Important Features and Functional Concepts*, Organization for the Advancement of Structured Information Standards (OASIS), Oct. 2004; http://uddi.org/pubs/uddi-tech-wp.pdf.
5. M. Jeronimo and J. Weast, *UPnP Design by Example*, Intel Press, 2003.
6. K. Arnold et al., *The Jini Specification*, Addison-Wesley, 1999.
7. *Jini Technology Surrogate Architecture Specification*, Sun Microsystems, 2003; http://surrogate.jini.org/sa.pdf.
8. *Specification of the Bluetooth System*, v. 1.1 core, Bluetooth Consortium, 2001; http://bluetooth.com.
9. E. Guttman, "Service Location Protocol: Automatic Discovery of IP Network Services," *IEEE Internet Computing*, vol. 3, no. 4, 1999, pp. 71–80.
10. T. Howes, *The String Representation of LDAP Search Filters*, IETF RFC 2254, Dec. 1997; www.rfc-editor.org/rfc/rfc2254.txt.
11. *Salutation Architecture Specification*, v. 2.1, Salutation Consortium, 1999.
12. W. Adjie-Winoto et al., "The Design and tion of an Intentional Naming System," *Proc. Symp. Operating Systems and Principles*, ACM Press, 1999, pp. 186–201.
13. S. Czerwinski et al., "An Architecture for a Secure Service Discovery Service," *Proc. 5th Ann. ACM/IEEE Int'l Conf. Mobile Computing and Networking* (MobiCom 99), ACM Press, 1999, pp. 24–35.
14. T. Kindberg and J. Barton, "A Web-Based Nomadic Computing System," *Computer Networks*, vol. 35, no. 4, 2001, pp. 443–456.
15. R. Want et al., "Bridging Physical and Virtual Worlds with Electronic Tags," *Proc. SIGCHI Conf. Human Factors in Computing Systems* (CHI 99), ACM Press, 1999, pp. 370–377.
16. A. Voida et al., "Listening In: Practices Surrounding iTunes Music Sharing," *Proc. SIGCHI Conf. Human Factors in Computing Systems* (CHI 05), ACM Press, 2005, pp. 191–200.
17. W.K. Edwards et al., "Challenge: Recombinant Computing and the Speakeasy Approach," Proc. 8th Int'l Conf. Mobile Computing and Networking (MobiCom 02), ACM Press, 2002, pp. 279–286.
18. B. Johanson, A. Fox, and T. Winograd, "The Interactive Workspaces Project: Experiences with Ubiquitous Computing Rooms," *IEEE Pervasive Computing*, vol. 1, no. 2, 2002, pp. 67–74.

the AUTHOR



W. Keith Edwards is an associate professor at the Georgia Tech College of Computing, where his research takes a human-computer interaction approach to networking and security problems. He received his PhD in computer science from Georgia Tech. He's a member of the ACM. Contact him at the College of Computing, Georgia Inst. of Technology, Atlanta, GA 30332-0760; keith@cc.gatech.edu.