Discovery and Its Discontents:
Discovery Protocols for Ubiquitous Computing

Presented at Center for Excellence in Space Data and Information Science
NASA Goddard Space Flight Center

April 5, 2000

Robert E. McGrath[1]
(mcgrath@ncsa.uiuc.edu)
National Center for Supercomputing Applications
Department of Computer Science
University of Illinois, Urbana-Champaign

[1] This work was partly supported by the National Center for Supercomputing Applications, which is funded by the National Science Foundation, the State of Illinois, and public and private partners. This paper comes from a collaboration with the professors and students of the 2K' operating system project, in the Department of Computer Science, University of Illinois, Urbana-Champaign. (http://choices.cs.uiuc.edu/2k)

Discovery and Its Discontents:
Discovery Protocols for Ubiquitous Computing

Robert E. McGrath
March 25, 2000

Introduction

Recent trends in mobile and ubiquitous computing have created new requirements for automatic configuration and reconfiguration of network devices. Furthermore, the exploding deployment of networked digital devices in diverse "real world" environments— including "hostile" environments such as homes— has increased the need to simplify network administration for all networks. In response to these requirements, a variety of new protocols have been proposed, which attempt to provide automatic "discovery" and configuration of network devices and services.

Some Definitions

The terminology in this area is not standardized. There are four key concepts, which may appear under different names.

- Device
- Service
- Lookup
- Discovery

Entities: "Devices" and "Services"

Devices and Services are the entities that participate in "discovery". "Devices" includes conventional computers, small hand-held computers (PDAs), and more specialized network devices, such as digital cameras, printers, telephones, etc. "Services" includes any sort of network service that might be available.

Part of the confusion lies in the fact that most devices are represented on the network by one or more "services", blurring the distinction to the point of non-existence. Furthermore, a single network attached device may implement several services, e.g., a network printer may provide printing and fax (and who knows what else), all in a single "device". For purposes of this paper, devices and services can be considered as essentially equivalent, in that any protocol that works for one will work for the other. Together, these will be termed "entities" or "resources" on the network.

"Lookup" and "Discovery"

I use the term "lookup" to refer to the process of locating a specific object, resource, or whatever. Lookup may be by exact name or address, or by some matching criteria. Lookup is "passive", in that it is initiated by a seeker, and requires the existence of some directory or other agent to answer the request. Lookup may be done in a statically configured environment , the directory need not be "writable". DNS [22] , LDAP [34] , and CORBA Naming Service[24] are all "lookup" services, as are countless other registries, directory services, and name services.

By contrast, "discovery" is used to refer to a more spontaneous process, in which many entities— not just the directory services— "discover" the other entities on the network, and present themselves to other entities. So-called "discovery" protocols have the overall goal of making digital networks easier to create and use. A discovery service may be used for lookup, but many "lookup" services do not support discovery. The most important features a discovery protocol are:

- "Spontaneous" discovery and configuration of network devices and services

- Selection of specific types of service
- Low (preferably no) human administrative requirements
- Automatically adaptation to mobile and sporadic availability
- Interoperability across manufacturers and platforms

The classic Internet protocols, such as the Internet Domain Name Service (DNS) [22] , do not meet these requirements because:

- They use static databases/files of information
- They are required to be maintained by privileged administrators
- They do not guarantee the availability of the objects registered
- They have limited semantics for searching
- They do not generate events when resources register and unregister

Some directory services such as LDAP [9, 34] and CORBA Name and Trader Services [24] can be used for service announcement and requests, but do not themselves specify protocols for spontaneous discovery. As will be discussed below, discovery protocols can be built on top of these services. And, of course, the "discovery" services discussed below may be used for "lookup".

Discovery protocols must face significant scaling issues. They seek to work in networks ranging from a few devices/services (e.g., a home) to a large enterprise with hundreds of thousands of devices/services. The devices will include a heterogeneous assortment small, special purpose units, such as digital cameras and printers. Such devices must not be required to implement extremely large and complex protocols in order to participate.

Several types of interoperability issues are important. A key requirement is to provide open standards for manufacturers to build to. For example, a maker of digital cameras would like the camera to plug in to everybody's network, and to be able to cooperate with any available printer. The services and devices must interoperate with other entities without pre-existing knowledge—this is a key aspect of "spontaneous" configuration. Note that there are two issues here : how the device presents itself to the network and other devices, and how the device or service itself can figure out how to interface to other devices and services.

2. Example Protocols

I will consider a sample of existing and up-coming "discovery" protocols. For comparison, three "lookup" services are discussed, followed by five "discovery" protocols. Table 1 lists the protocols discussed and their sources.

Table 1. Lookup and Discovery Protocols

Protocol	Source
"Lookup"	
Domain Name Service (DNS)	IETF RFC [11, 22]
Lightweight Directory Access Protocol (LDAP)	IETF RFC 2251 [34]
CORBA Trader Service	Object Management Group (OMG) [24]
"Discovery"	
Salutation	Salutation Consortium [30]
Service Location Protocol (SLP)	Sun, IETF RFC 2608 [14]
JINI	Sun/JavaSoft [6]
UPnP + Simple Service Discovery Protocol (SSDP)	Microsoft [20] , IETF Draft [10]
Secure Service Discovery Service (SSDS)	UCB Ninja Project [23]

2.1 Lookup (Directory) Services

As discussed above, there are many "lookup" services. I will briefly discuss three of these to show how they do not meet the needs for "discovery".

Domain Name Service (DNS)

The Internet DNS is used for discovery of addresses for network devices and services, but is poorly suited for spontaneous discovery. [22] The classic DNS protocol provides a static database of name-address maps, which are maintained by privileged users. Recent extensions to DNS support a very limited set of service types and a few attributes that can be used to search.[11] It is difficult to extend the types of information in a DNS service, and arbitrary user applications may not add or modify the DNS database.

DNS is organized as a hierarchy of servers, in the familiar Internet domain names. This scheme has been shown to scale up to the entire Internet.

DNS is a trusted service, security is provided by controlling access to a few privileged users.

Lightweight Directory Access Protocol (LDAP)

LDAP is an IETF standard provides a scalable hierarchy of name spaces, through which services can advertise and clients can locate services. [34] LDAP is extensible, so many kinds of information can be served (e.g., see the Globus "Metacomputing Directory Service" [9]). LDAP's query model is adequate for many kinds of discovery protocols. However, LDAP itself does not specify protocols for "spontaneous" discovery : it does not multicast announcements, nor check the existence of registered services. These features can be built on top of LDAP, e.g., see the SLP below.

LDAP is organized as a hierarchy, and the queries can be limited to particular parts of the hierarchy. This design scales up to the same scale as DNS, although queries over very large domains are likely to be very inefficient.

Despite its name, LDAP is a "heavy" protocol, which would be less than ideal for implementation on a small device. Also, partly because of the complexity, it is not clear that LDAP is well suited for either near real-time discovery, or for very large numbers of services.

LDAP relies on other network services for security.

CORBA Trader Service

The CORBA Trader Service is a standard CORBA Service. [24] The Trader Service (TS) provides an interface for advertisement of services and for service requests by attributes. Any device or service that has a CORBA implementation, proxy, or wrapper can use the Trader Service to advertise its attributes. CORBA objects can use the Trader Service to locate objects that meet particular constraints.

The TS is defined as CORBA interfaces, and all advertisements, requests, and replies are CORBA objects. Requests may be framed as constraint expressions, which are constructed as boolean and arithmetic expressions with values of CORBA variables.

The TS does not guarantee that the registered objects are available, and it does not provide notification. These features can be implemented fairly easily, using CORBA Event Service and other standard features of CORBA.

The CORBA TS supports federation of trader services into a single logical service. The federation may have an arbitrary topology, not limited to a hierarchical tree. Federation should allow considerable scalability, depending on the details of the implementation.

The TS can use CORBA security, which can be very strong if implemented. The TS itself has no security framework of its own.

The TS obviously depends on CORBA, all participants must be cast as CORBA objects and use CORBA protocols. It may be difficult to implement the required protocols on a small device. However, CORBA is a very effective platform for creating proxies which run on other systems, and it is comparatively easy to create CORBA objects to represent small devices. [28]

2.2 Discovery Protocols

"Discovery" protocols address some of the missing features of lookup services, especially features to support "spontaneous" and low overhead network configuration.

Salutation

The Salutation protocol is an open specification that provides "spontaneous" configuration of network devices and services. Salutation is already in use by a consortium of companies that make printers and similar devices, including HP, IBM, Xerox, and AOL. [25, 30]

The Salutation architecture defines an abstract model with three components: Client, Server, and Salutation Manager (SLM). The Salutation Manager manages all communication, and bridges across different communication media as needed. Salutation defines its protocol based on SunRPC. [31]

The model can be implemented with or without a separate directory service (SLM), and directories (SLMs) can be organized as a hierarchy or other graph of cooperating directories. When implemented without a directory, clients and services can locate each other directly, using local broadcast. This "directory-less" configuration allows Salutation to work correctly on a network with no administration at all, e.g., in a home or automobile.

Salutation defines a specific (extensible) record format for describing and locating services. This format includes service type (such as ["PRINT"]) and attributes (such as "color"). Services advertise by registering with one or more Salutation Managers. Clients locate services by sending service requests. These may include:

- List all services
- List all services of particular type
- List services that match specific attributes (custom matching functions can be registered)

The registry returns the address and a "Personality Profile", which is a description of the service and its interface. The profiles are specified in great detail (e.g. [32] which is 275 pages!).

The use of SunRPC seems to place limits on Salutation. The only multicast available is broadcast RPC. This is used to discover local instances of the Salutation Manager. Also, the security is very weak, since SunRPC doesn't provide strong crypto support.

Except for SunRPC, Salutation is technology neutral. It is clear that Salutation can also be implemented with LDAP or any number of other directory services. Salutation is also designed for and highly compatible with wireless technologies, there are already Salutation bindings for IrDA [26] and Bluetooth [21] .

Service Location Protocol (SLP)

SLP comes from Sun Microsystems, and is an IETF standard for "spontaneous" discovery of services. [12, 14] SLP defines an abstract architecture consisting of "User Agents" (UA) (clients), "Service Agents" (SA) (services) and "Directory Agents" (DA) (directories). The UA perform service discovery on behalf of clients, the SA advertise the location and attributes of services, and the DS aggregates service information. Multiple DAs can be used for replication or to provide a hierarchy or graph of domains.

The SLP can be implemented in several configurations. In "passive" configuration, the DAs periodically multicast service advertisements. CAs and SAs can also locate DAs using DHCP. [27] SLP can be implemented without any DA at all, enabling SLP to work with no administration. In the absence of a DA, the UAs and SAs implement all the functions of the DA with multicasts. When one or more DA is present, the protocol is more efficient, as the CA or SA uses unicast messages to the DA.

The SLP defines a "Service URL" [14] , which encodes the address, type, and attributes of the service. For example,

service:printer:lpr://hostname

Might be the service URL for a line printer service. Service requests may match according to service type or by attributes. Attribute matching is specified by a template [13] and an LDAPv3 predicate. [34] This is a fairly powerful syntax for matching.

SLP requires UDP/IP multicasts, and may use TCP for exchanging large messages. SLP uses multicast carefully, keeping track of known recipients to minimize retransmissions. SLP can be configured in different ways to trade network traffic against currency of the globally known state. With fewer multicasts, the system converges to the correct global state more slowly, while faster convergence is assured at the cost of much greater network usage.

SLP is designed to work well with LDAP, but does not require LDAP.

SLP provides for authentication, but does not specify it. Similarly, encryption is left to other protocols.

JINI

Sun's JINI is a Java environment which supports spontaneous discovery. [6] In many ways, JINI resembles SLP, and was clearly influenced by it. However, JINI is very tightly bound to the Java environment. The protocol is mostly defined as exchanges of serialized Java objects, mostly via Java Remote Method Invocation (RMI). [33]

JINI requires at least one copy of the "JINI Lookup Service" (LS), which has a standard Java interface. A Java client program begins "discovery" with a multicast UDP/IP to locate instances of the lookup service. Alternatively, the Lookup Server may multicast announcements of its availability. Either way, once one or more LS successfully, the client or service will obtain an RMI stub to access the JINI Lookup Service. All other service advertisement and location is done via the Lookup Service. In each case, the protocol is done by exchanging serialized Java objects via Java RMI.

Servers advertise by registering a Java RMI stub with the JINI Lookup Service. Clients locate services by requesting specific types of service. The request is basically a simple template for matching string attributes. However, the request and the matching must be implemented as Java objects, following JINI specified interfaces. Despite the use of Java objects, the JINI filtering mechanism is far more limited and less powerful than LDAP (and SLP), CORBA Trader Service, and the XML approaches described below.

When a service is located, JINI delivers a Java RMI stub to access it. This allows clients to load code at run time, and allows services to "push" their interfaces to clients they have never met. This is clearly a powerful feature, made possible by the single language environment and the mobility of Java code.

The JINI protocol is based on leases, all advertisements and registrations are for a specific and fairly short period of time. Long running clients and services must renew their leases periodically, entities that crash are automatically removed from all lookup services when the lease expires. Leases assure that JINI recovers from crashed entities, and periodic renewal of leases by entities rebuilds the global state in case of a Lookup Server failure.

The JINI LS also provides notification of the arrival and departure of entities, and also notification to the entity itself when it is "discovered". These notifications enable the run time composition of services, and the construction of sophisticated services to manage a dynamic environment. Together, the responsiveness of notification and "self-healing" of leases can potentially make a JINI system very robust without manual intervention.

JINI does not support "directoryless" operation, and cannot interoperate with any other protocol or language environment. Since the protocol depends on the use of Java stubs, a device must either implement a JVM, or else use a proxy. These requirements make JINI less than ideal for use with very small devices— despite Sun's aspirations.

JINI Lookup Services can be federated in arbitrary topologies. In theory, this should allow construction of large systems.

As JINI is based on Java, it provides the same weak security as Java provides. Neither the "discovery" nor the JINI Lookup Server provide mandatory strong cryptography. In theory, JINI can be partly secured by creating custom Java Virtual Machine (JVM) and/or "SecurityManager" classes, but full security would seem to require a rewrite of RMI (e.g., see below, the Ninja SSDP.)

Universal Plug and Play (UPnP) and Simple Service Discovery Protocol (SSDP)

UPnP is a Microsoft standard for spontaneous configuration. [4, 19, 20] UPnP handles network address resolution, and coupled with the IETF proposal Simple Service Discovery Protocol (SSDP) [7] it provides higher level service discovery. UPnP has a similar architecture to Salutation and SLP, and was no doubt influenced by (and intended to compete with) them. UPnP uses XML for device/service description and queries, which brings it into the mainstream of the evolving WWW.

As in JINI, UPnP has a multi-stage protocol. At the base, UPnP provides "simple discovery", in which network addresses are discovered. Advertisement is done by a local broadcast announcement. When successful, "simple discovery" returns an IP address or URL plus a "device type". Services are described by extended URLs, similar to (but completely incompatible with) SLP. The URL is for an XML file with an elaborate description of the device. Starting with this URL, the SSDP defines a Web based discovery protocol, which uses HTTP (with extensions). While HTTP is obviously a very heavy weight protocol, it is claimed that the necessary subset can be implemented with a small foot print. ([4] , p. 7).

A UPnP "device" is said to export one or more "services". Services are describe in XML, and the XML can be a complete abstract description of the type of service, the interface to a specific instance of the service, and even the on-going (virtual) state of the service. The interface and state descriptions are intended to allow clients to implement custom interfaces to devices, by mapping local displays and operations to the abstract state and interface represented in the XML.

The XML description can be used by programs or browsers to locate specific services by filtering on XML tags or combination of tags. XML is extremely flexible, so it can deliver almost any kind of information (including "essentials" such as corporate logos and warrantee information). These descriptions go far beyond the information available from SLP/LDAP, Salutation, or JINI.

UPnP requires IP, not to mention HTTP and XML. Non-IP networks and interconnects can be bridged, at least at the level of the XML, if not elsewhere. UPnP has no specific security features. It depends on the network and Web infrastructure for its security. Thus, security is clearly "optional".

UPnP can work with no central directory of addresses, but clearly the full XML capability requires a Web server somewhere.

UCB Ninja: Secure Service Discovery Service (SSDS)

The SSDS is part of the University of California, Berkeley Ninja research project. [5, 23] The SSDS is similar to other discovery protocols, with a number of specific improvements in reliability, scalability, and security. Although SSDS is implement in and relies on Java, it uses XML for service description and location, rather than Java objects.

The SSDS model has Clients, Services, and Secure Discovery Service (SDS) Servers. SDS Service availability is announced by periodic (authenticated) multicasts from the SDS Server. The multicast message contains URLs for the available SDS server. The SDS server (and clients) may cache service information, but the state of the system can be constructed entirely from the multicasts. This scheme provides scalability, error recovery, and self-healing, which can make the system extremely robust with minimal manual intervention.

The SSDS is implemented in the form of Java RMI remote methods. The protocol is designed as an exchange of XML "documents", and service location is done by matching of XML tags. This is logically equivalent to UPnP, and the two flavors of XML should theoretically interoperate by automatic mapping. It is argued that other forms of service advertisements, including JINI objects, can be translated into XML as well.([5] , p. 33)

The Ninja project has explored the possibility of automatically mapping interfaces to each other using XML descriptions. For example, a generic controller client might be able to map its controls to new devices by algorithmically mapping the description of the device (XML) to the description of the controller (XML). [5, 15] If this can be implemented, it will be extremely important for "spontaneous" networking and interoperability of heterogeneous devices.

An important distinction of the SSDS provides extremely strong mandatory security: all parties are authenticated, and all message traffic is encrypted. The SSDS uses several kind of authentication, for different purposes. Authenticated and encrypted communication is provided by a custom re-implementation of the RMI protocol. [5, 35] Ninja security is discussed further in section 3, below.

The SDS Servers are organized as a hierarchy, to which additional servers can be dynamically added to scale up the system under heavy load. The hierarchy of servers also detects and restarts a failed server.

The Ninja project has strongly influenced IBM "Universal Information Appliance (UIA)" project. [7]

3. Discussion and Conclusions

Many Common Features

Given the similar requirements, all these protocols provide similar basic services. At base, "clients"(consumers) must find relevant "services"(providers), including sufficient information to establish contact and obtain service. There are two basic mechanisms (design patterns?) by which this is accomplished: advertisement and service request. Services "advertise"their availability, address, and other necessary information. Clients who receive advertisements may then contact services as they wish. Alternatively, clients may "request"service of some kind, and receive information about services in response. Services or other agents listen for requests and respond appropriately. Each protocol implements one or both of these concepts, although the details differ considerably.

All of these protocols support some kind of hierarchical or federated organization, which can be used to improve reliability through redundancy, and improve scalability through distribution. Despite some marketing claims, none of these protocols can scale up to the whole Internet. It is not clear that this is even a reasonable goal (i.e., is it reasonable to expect a PDA to "simply work"on any network on the planet?). However, most of these discovery protocols will work on large and highly dispersed networks, which will meet the most important needs.

Is A Directory Service Required?

Discovery is essentially a problem of determining the global state of a decentralized system. In the face of failures and arbitrary delays, the global state of a real distributed system can only be approximately known. The goal must be to provide a good enough approximation with reasonable response time and low overhead.

One key architectural issue is the use of one or more "directories". One way to implement service location is to provide a directory (or registry or lookup) service, which provides a database for all services. Advertisement is implemented by registering with the directory, clients may poll information and/or receive notifications from the directory service. This approach can be scaled up by replicating or otherwise distributing the directory service.

Directories have the disadvantage that they require at least some administration. If discovery requires the directory service, then the directory service must be present on all networks at all times. This tends to reduce the possibility of zero-overhead configuration, e.g., a home network in which the video camera and player "simply work". Some protocols support directory-less operation, in which all the participants can advertise and/or request service from each other. In this case, the protocol must be kept very simple, so that simple devices with very limited resources can implement it.

Network Usage

The discovery service may make a variety of assumptions about the network infrastructure. Many services use IP addresses and UDP datagrams. While using IP has advantages and covers a very large range of systems, requiring IP is a serious limitation for very small devices, and for wireless technology such as IrDA [18] and Bluetooth [2] . In these cases, it may be necessary to provide a proxy to bridge the protocol— which is very undesirable overhead for the lowest end configurations.

Parsimonious use of bandwidth is an important goal for discovery protocols. Naïve protocols (e.g., "each services advertise once per second") can easily saturate the network. One of the strong advantages of a directory service is that communication can be restricted to a minimum: almost all discovery-related traffic can be point-to-point communications with a local copy of the directory service. (E.g., JINI [6])

Hierarchy or federation can be used to try to localize the traffic and reduce the number of parties involved. However, for any given network, it is important to manage the bandwidth needed for discovery protocols. This is especially important for low cost, low bandwidth networks such as a home network or a wireless device.

Simple analysis shows that a purely advertisement or purely request strategy may use a great deal of bandwidth. (See [12] for a good discussion of this.) For example, if all services must advertise their availability periodically, the network may fill with advertisements, even if nothing has changed. Similarly, if each client must send service requests to all possible services, the network will fill with requests and the client may be swamped with responses. Clearly, the most efficient protocol would maintain the current state and require messages only when a service or client changes. Furthermore, careful use of multicast can limit the number of messages required to maintain this state. [12]

Service Description and Filtering

One of the key goals of "discovery" is to locate instances of specifically desired services. For instance, a digital camera needs to find a printer that is nearby, with specific attributes, e.g. full color and 600 DPI. To support this, a discovery service should provide "filtering" for service requests. The greater the number and variety of services available, the more important the filtering will become.

There are, of course, a great variety of possible approaches to filtering. It should be realized that each of these amount to concrete realizations of a conceptual model of what kinds of devices and service may exist, and what clients may ask for.

A closely related question is, when a service is identified to a client, what does the client receive? For "spontaneous" configuration to succeed, there is a bootstrap problem: the server and client must share enough common semantics to establish communication and negotiate appropriate interfaces and parameters. This semantics is essentially the same conceptual model of the devices, services, and their attributes noted above.

Designing and implementing the meta-protocol of service types, etc., is an especially difficult problem for open, dynamic networks, in which the communicating parties may have never encountered each other before, and cannot assume shared code or architectures.

It has become clear that XML is the technology of choice for this task. XML is general enough to express the required concepts, it is rigorously specified, and it is universally accepted and deployed. Furthermore, XML is specifically designed to support automatic translation and transformation between XML languages. This provides a critical capability for interoperating multiple services and devices.

While XML may solve the problem of delivering service descriptions, it cannot address the fundamental conceptual issues. It is still necessary (and very difficult) to design conceptual models and maps between different models.

Security

Discovery protocols face a real challenge from security. First, the desire to be automatic, lightweight, and to minimize network usage forces protocols to use very simple schemes. It is important that the digital camera "simply work" without elaborate protocols for establishing keys, passwords, etc. On the other hand, security is definitely needed : my digital camera must not be able to use a printer in the neighbor's house without permission, and the pictures from my camera must not be intercepted by unauthorized parties on the way to my printer.

A second problem is definition of appropriate security models and policies. What are appropriate models of "trust" for spontaneous networks? What sort of access control should a discovery service provide? The Ninja project has done a good job on this, and they use several different security protocols and services, in an eclectic mix [5] :

- Public Key authenticated SDS server announcements
  - *Assures authenticity of discovery service*
- One-way encrypted service announcements (combined public and private key protocol)
  - *Assures privacy and authenticity of service descriptions*
- Secure RMI
  - *Two-way authenticated and encrypted remote method invocation*
- Certificates and a Certificate Authority structure
  - *Capabilities to authenticate all principals*

The Ninja work suggests that existing security mechanisms are probably sufficient, but protocols for their use need to be investigated and implemented.

Scope of Discovery: Geographical and spatial scoping

All the discovery protocols discussed here are "administratively scoped", i.e., they locate services and devices within an administratively defined network domain. These domains are logical, and need not correspond to the physical environment. "Nearby" means nearby within the network topology, not physical space.

This virtuality is one of the great strengths of the Internet and related technology. However, it is often necessary to locate physically (not logically) near-by resources. For instance, if one needs to view a screen, it is important to find a display that is within easy visibility of the user, not one that is "close" to the client or server software in the network topology. Another example is a location specific "help page", which has a well-known network address, but whose content is customized to the physical location of the receiver.

Providing physical location can be done in two ways:

1. Network devices and services can "know" their location, and can detect what and who is near them.
2. Devices and people can "detect" their own location and report to services.

In the first case, some components of the system are charged with tracking the location of everything within their scope. In the second case, devices are charged with determining their own location and reporting it to the system. A real system might well use both methods.

One way to provide awareness of physical locality is to provide a standard service to detect the presence of people and devices. Note that this service requires both mechanisms for detecting arrival and departure (IR badges, radio beacons, video and sonar detectors) and also policies about who and what is to be tracked, and under what circumstances an entity is to be admitted.

This service could be the standard discovery service, or a separate service that interacts with the discovery protocols. For example, HP's CoolTown implements an "Inventory Manager" service that is separate from their discovery service. [3]

The other alternative is to have devices detect their own position. This might be done through beacons (e.g., the Georgia Tech "Cyberguide" [1]) or GPS (e.g., [8, 29]). While each device knows its own location, communication is required to establish the location of other entities to establish the overall context.

Another useful approach would be spatially limited multicasts, e.g., as proposed by the DataMan project at Rutgers. [16, 17] Conventional multicasts are limited to administrative and other

network bounds, e.g., hop counts. These limits have no necessary relation to physical space, so there is no way to route a multicast to a spatial region, e.g., to a single room or floor of a building. Spatially addressed multicasts would allow mobile devices to tune to well-known multicast channels that carry correct local service. For example, the address of the relevant discovery service could be broadcast on the same channel to each spatial zone, with only the correct messages arriving in the appropriate locations.

In addition to "awareness" of location, protocols need to recognize and provide semantics for spatially limited or constrained lookups, and possibly constraints based on geometry of the physical space. For example, it is important to be able to request a display screen that is "near" the user so it is visible. Visibility cannot be determined from the location of the server(s), nor even the GPS position of the person and display (if known). "Visibility" is a geometry problem, and both constraint language and service attributes will need to implement models of spatial geometry. This is a very difficult problem.

Interoperability

A very obvious and unfortunate conclusion of this survey is that there are far too many "standards" at this time. Many of these protocols are logically compatible, and can be fairly easily mapped and bridged. Unfortunately, "discovery" really should be universal, and it shouldn't be necessary to implement an array of equivalent protocols, or to have multi-protocol proxies. Worse, small, dedicated information appliances are specifically not supposed to contain this kind of complex multi-purpose code.

The reasons for this undesirable diversity include:

- the comparative novelty of the application and market
- inherent technical challenges
- a spirit of experimentation
- jockeying for market share

It is unlikely this diversity will continue for long. For mass production, manufacturers need to build to a single standard. It is difficult to know which approaches will prevail, and at a technical level it doesn't matter very much. It is likely that this will be a case where market share rather than technical merit will decide.

References

1. Abowd, Gregory D., Atkeson, Christopher G., Hong, Jason, Long, Sue, Kooper, Rob, and Pikerton, Mike, "Cyberguide: A mobile context-aware tour guide," *ACM Wireless Networks*, vol. 3, no. 5, pp. 421-433, 1997. http://www.acm.org/pubs/contents/journals/wireless/1997-3-5/p421-abowd.pdf

2. Bluetooth Consortium, "The Bluetooth Consortium," http://www.bluetooth.com

3. Caswell, Deborah, "Creating a Web Representation for Places," *Proceedings of the Ninth International World Wide Web Conference (Submitted)*, 2000. http://cooltown.hp.com/papers/PlaceManagerv4.htm

4. Christensson, Bengt and Larsson, Olof, "Universal Plug and Play Connects Smart Devices," *WinHEC 99*, 1999. http://www.axis.com/products/documentation/UPnP.doc

5. Czerwinski, Steven E., Zhao, Ben Y., Hodes, Todd D., Joseph, Anthony D., and Katz, Randy H., "An Architecture for a Secure Service Discovery Service," *Mobicom'99*, 1999. http://ninja.cs.berkeley.edu/dist/papers/sds-mobicom.pdf

6. Edwards, W. Keith, *Core JINI*. Upper Saddle River, NJ: Prentice Hall, 1999.

7. Eustice, K. F., Lehman, T. J., Morales, A., Munson, M. C., Edlund, S., and Guillen, M., "A universal information appliance," *IBM Systems Journal*, vol. 38, no. 4, , 1999. http://www.research.ibm.com/journals/sj/384/eustice.html

8. Feiner, Steven, MacIntyre, Blair, Hollerer, Tobias, and Webstar, Anthony, "A Touring Machine : Prototype 3D Mobile Augmented Reality Systems for Exploring the Urban Environment," *First International Symposium on Wearable Computers (ISCW'97)* Cambridge, MA, 1997.

9. Fitzgerald, Steven, Foster, Ian, Kesselman, Carl, Laszewski, Gregor von, Smith, Warren, and Tuecke, Steven, "A Directory Service for Configuring High-Performance Distributed Computations," *The 6th IEEE Symposium on High-Performance Distributed Computing* 1997. ftp://ftp.globus.org/pub/globus/papers/hpdc97-mds.pdf

10. Goland, Yaron Y., Cai, Ting, Leach, Paul, Gu, Ye, and Albright, Shivaun, "Simple Service Discovery Protocol,"IETF, Draft draft-cai-ssdp-v1-03, October 28 1999. http://search.ietf.org/internet-drafts/draft-cai-ssdp-v1-03.txt

11. Gulbrandsen, A. and Vixie, P., "A DNS RR for Specifying the Location of Services (DNS SRV),"IETF RFC 2502, October 1996. http://www.rfc-editor.org/rfc/rfc2052.txt

12. Guttman, Erik, "Service Location Protocol : Automatic Discovery of IP Network Services," *IEEE Internet Computing*, vol. 3, no. 4, pp. 71-80, 1999. http://computer.org/internet/

13. Guttman, E., Perkins, C., and Kempf, J., "Service Templates and Service: Schemes,"IETF, RFC 2609, June 1999. http://www.rfc-editor.org/rfc/rfc2609.txt

14. Guttman, E., Perkins, C., Veizades, J., and Day, M., "Service Location Protocol, Version 2," IETF, RFC 2608, June 1999. http://www.rfc-editor.org/rfc/rfc2608.txt

15. Hodes, Todd and Katz, Randy H., "A Document-based Framework for Internet Application Control," *Second USENIX Symposium on Internet Technologies and Systems*, Boulder, CO, 1999. http://daedalus.cs.berkeley.edu/publications/docu-usits99.ps.gz

16. Imielinski, Tomasz and Badrinath, B. R., "Mobile Wireless Computing," *Communication of the ACM*, vol. 37, no. 10, pp. 18-28, 1994.

17. Imielinski, Tomasz and Navas, Julio C., "GPS-Based Geographic Addressing and Routing," Rutgers LCSR-TR-262, 1996.

18. IrDA, "Technical Summary of "IrDA DATA" and "IrDA Control"," http://www.irda.org/standards/stantards.asc

19. Microsoft Corporation, "Universal Plug and Play Device Architecture Reference Specification,"Microsoft Corporation November 10 1999. http://www.microsoft.com/hwdev/UPnP

20. Microsoft Corporation, "Universal Plug and Play : Background," http://www.upnp.org/resources/UpnPbkgnd.htm

21. Miller, Brent, "Mapping Salutation Architecture APIs to Bluetooth Service Discovery Layer," Bluetooth Consortium 1.C.118/1.0, 01 July 1999.

22. Mockapetris, P., "Domain Names-Implementation and Specification,"IETF RFC 1035, October 1987. http://www.rfc-editor.org/rfc/rfc1035.txt

23. Ninja, "The Ninja Project," http://ninja.cs.berkeley.edu

24. Object Management Group, "CORBAservices: Common Object Services Specification," Object Management Group 1999. ftp://ftp.omg.org/pub/.docs/formal/98-07-05.pdf

25. Pascoe, Bob, "Salutation Architectures and the newly defined service discovery protocols from Microsoft and Sun,"Salutation Consortium, White Paper June 6 1999. http://www.salutation.org/whitepaper/JINI-UPnP

26. Pascoe, Bob, "Salutation- Lite,"The Salutation Consortium June 6 1999. http://www.salutation.org/whitepaper/Sal-Lite.PDF

27. Perkins, C. and Guttman, E., "DHCP Options for Service Location Protocol,"IETF RFC 2610, June 1999. http://www.rfc-editor.org/rfc/rfc2610.txt

28. Roman, Manuel, Singhai, Ashish, Carvalho, Dulcineia, Hess, Christopher, and Campbell, Roy H., "Integrating PDAs into Distributed Systems: 2K and PalmORB," *International Symposium on Handheld and Ubiquitous Computing (HUC'99)* Karlsruhe, GE, 1999. http://choices.cs.uiuc.edu/2k/papers/huc_99_ps.gz

29. Ryan, Nick, Pascoe, Jason, and Morse, David R., "FieldNote : a Handheld Information System for the Field," *First International Workshop on TeloGeoProcessing (Telegeo'99)* Lyon, 1999. http://www.cs.ukc.ac.uk/research/infosys/mobicomp/Fieldwork/Papers/TeleGeo99/TeleGeo.ps

30. Salutation Consortium, "Salutation," http://www.salutation.org

31. Salutation Consortium, "Salutation Architecture Specification (Part-1) Version 2.1,"The Salutation Consortium 1999. http://www.salutation.org

32. Salutation Consortium, "Salutation Architecture Specification (Part-2),"The Salutation Consortium 1999. http://www.salutation.org

33. Sun Microsystems, "Java Remote Method Invocation (RMI),"
http://java.sun.com/products/jdk/1.2/guide/rmi/index.html

34. Wahl, M., Howes, T., and Kille, S., "Lightweight Directory Access Protocol (v3),"IETF RFC 2251, December 1997. http://www.rfc-editor.org/rfc/rfc2251.txt

35. Welsh, Matt, "Ninja RMI: A Free Java RMI,"
http://www.cs.berkeley.edu/~mdw/proj/ninja/ninjarmi.html