

Naming and Discovery in Mobile Systems

Guanling Chen,[†] Kazuhiro Minami,[‡] David Kotz[‡]

[†]Department of Computer Science, University of Massachusetts Lowell
glchen@cs.uml.edu

[‡]Department of Computer Science, Dartmouth College
{minami, dfk}@cs.dartmouth.edu

Abstract

Middleware supporting mobile applications must provide naming and discovery functionalities to enable anytime and anywhere service access. In this chapter, we survey existing service-discovery standards, identify four challenges for naming and discovery in a mobile environment, and provide a detailed discussion of the approaches that can be used to address each of these challenges.

Contents

1 Introduction	1
1.1 A general model	2
1.2 Challenges for a mobile environment	4
2 Existing standards	4
3 Mobility	6
4 Scalability	8
5 Context Awareness	10
6 Security and Privacy	12
7 Summary and Future Work	16

1 Introduction

Much of the technology necessary to realize Mark Weiser’s vision of ubiquitous computing [37] is now available. Small portable devices, and the wireless networks to support them, are pervasive. Sensors capable of location tracking [23] and environmental monitoring [24] will soon be small, cheap, and plentiful. The resulting mobile and pervasive-computing environments are crowded, heterogeneous, and always changing. To succeed without distracting the user, middleware systems must provide naming and discovery methods to facilitate anytime and anywhere service access.

Let us first consider a simple scenario to help understand the problem at hand. Suppose Alice is visiting an university to meet several of her colleagues, and she needs directions to various offices in different buildings. The campus is covered by an 802.11 wireless network and Alice can use her dual-mode mobile phone to gain access. For the whole campus there is a location service that can provide rough location information. For instance, it can tell which buildings Alice is nearby depending on which access point (AP) Alice's phone is currently associated with [26]. For individual buildings there may also be separate location services, based on extensive "radio fingerprints" maps that allow the service to return a precise location by comparing with the signal strength perceived by Alice's mobile phone [6]. As Alice moves around, the map application on her phone needs to find, select, and maintain communication to the appropriate location services depending on her current location. Alice should also be able to redirect the map to a nearby display or print it on a nearby printer for better readability. This scenario clearly illustrates the need for a middleware to bridge the services and clients, and the mobile environment poses several challenges we discuss below.

In this chapter we investigate the question of how the middleware can effectively and efficiently enable clients, such as Alice's mobile phone, to discover and interact with desired services, such as the location and display services mentioned above, in a mobile environment. As a framework for our discussion, we first give a general naming and discovery model in Section 1.1. We then identify the challenges for the mobile middleware to provide the naming and discovery functionalities in Section 1.2.

1.1 A general model

Here we define a general model for naming and discovery, as shown in Figure 1. In this model, the services *register* with the middleware while the clients *query* the middleware to get *results* (the desired services). The service registrations include a *name* describing the service properties and some other information. Although it is possible to provide type-only service discovery, we believe that service names are more expressive and necessary to distinguish services with the same type. A service may need to *update* its registration to reflect changes. A client's query could be persistent; once it is stored, the middleware will notify the client whenever the result of that query changes. All service names form a *name space* and a *directory* is a data structure used to store service names and other information. A middleware may contain a centralized directory storing the whole name space, or a set of distributed directories, each of which stores either the full name space or portion of it.

A name is a description of the service, which could be a simple string, a set of attributes, or a full-blown XML document [10]. A string may or may not have a syntactic structure, such as "printer120" or "/device/printer/120". A string representation typically has limited expressiveness, and the syntax could be

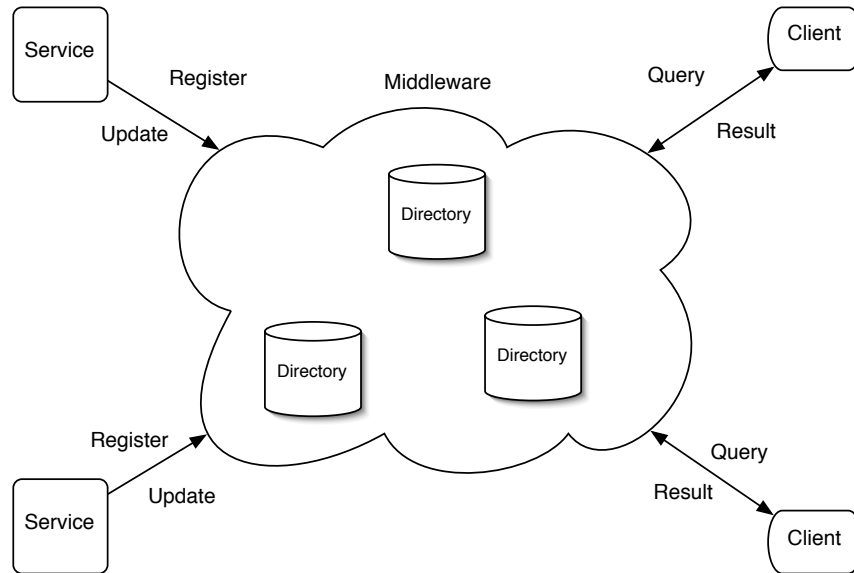


Figure 1: A general model for naming and discovery middleware.

awkward with a complex description. Another commonly used representation is attribute-based naming. An *attribute* is a key-value tuple and a name contains a set of attributes. Note that an attribute could be the child of another attribute, forming a hierarchy. For instance, the printer could be named as

```
[service=printer, color=true, building=sudikoff [floor=1 [room=120]]].
```

XML-based names are also expressive and the standard syntax facilitates the interoperability between services and clients. In addition to the properties of the service, XML can also be used to specify the syntax and semantics of the service’s functional interface, allowing a client to directly invoke these methods by inspecting the XML names. A disadvantage of XML-based naming is its verbose syntax, leading to increased processing overhead.

A query is typically specified in the similar syntax as the name encoding, using either a simple string, a set of attributes, or an XML description. Given a query, the middleware returns all names that *match* the query. The matching criteria could be simple equality tests, wildcard comparison, a subset of matching attributes, or a complex evaluation of XPath and XQuery for XML-based names.

Once the middleware returns the names that match a client’s query, the client can then directly communicate with the selected services. However, the communication could also be mediated through the middleware, in which case the middleware acts as a broker by passing the messages back and forth between the client and service. We discuss the tradeoffs between these two approaches later in the chapter. Note that “middleware” is a general term, and it could be either a distributed infrastructure, as shown in Figure 1,

or some simple library code embedded in services and clients so they can speak the same protocols. In this chapter, we are mainly concerned with an infrastructure-based middleware that contains one or more directories.

1.2 Challenges for a mobile environment

One goal of the middleware for naming and discovery is to provide interfaces or protocols to facilitate the interactions between clients and services with minimum administration and configuration overhead. There are four challenges that need to be explicitly addressed for a mobile environment.

The first problem to be addressed is the *mobility* of services and clients. Device movement may cause an update of service names, a change of communication location or addresses, change of the attachment points to the middleware, or weak network connections or temporary disconnections. For instance, Alice's phone may roam from AP to AP as Alice moves on campus. The communication between Alice's phone and the location service must be maintained, even when the phone's network address has changed.

The second challenge is *scalability*, that is, whether the middleware can scale beyond the local network to support a large number of services and clients while avoid excessive network traffic. For instance, thousands of devices on campus may query the middleware to find an appropriate location service, and the middleware must also support tens of thousands of devices in the environment, such as displays, printers, and sensors.

A third issue, often, is that the service registration and client query are *context-dependent*. Namely, the middleware must also support context monitoring to dynamically adapt naming and discovery processes by itself rather than leaving this burden to services and clients. For instance, the middleware must be able to return an appropriate location service that best resolves Alice's current location as she moves on the campus.

Finally, *security and privacy* must also be addressed by the mobile middleware. For instance, though Alice may allow her colleagues to view her current location, she may not want to reveal her location to the general public. There are many trust, authentication, and access control issues.

2 Existing standards

Jini is a Java-based architecture, introduced by Sun Microsystems in 1998, which enables a dynamic federation of users and resources [35]. Network services first use a *discovery* protocol to locate a Jini *lookup* service to which the services register using a *join* protocol. The lookup service then stores both the service attributes (describing the services) and the service objects (implementing appropriate interfaces). A client queries the Jini lookup service and downloads the service object matching the query. The service object acts

as the proxy between the client and the service provider, and this interaction does not go through the lookup service. So one advantage of Jini is that the proxy service object is downloaded to the client on the fly, and there is no “device driver” to be manually installed for service interaction. Both service registration with lookup and client access to Jini services leverage *leases*. The lease must be renewed before its expiration for continuous registration and access. The lease-based access control gives Jini another advantage: it is robust against abrupt device failures and departures. Jini also supports remote *events* to notify the clients about the changes in the system, such as arrival of new services or state change of existing services.

Universal Plug and Play (UPnP), extending Plug and Play (PnP) by Microsoft, aims at automatic discovery and control of devices [34]. UPnP uses Simple Service Discovery Protocol (SSDP) for service discovery, which is similar to the discovery, join, and lookup protocols used by Jini. The counterpart of Jini’s lookup service in UPnP is called *control point*. Unlike Jini, however, UPnP leverages standard TCP/IP and HTTP protocols for communication and XML for service description, eliminating the need of relying on a single programming language. The XML-based description is much richer than Jini’s simple attributes, and it contains a URL to which clients can send their control messages (also encoded as XML). Note that this approach removes the need to install any service-specific code on the client. UPnP further reduces administrative overhead by enabling devices to obtain an IP address automatically through AutoIP, when DHCP is not available, without explicit network configuration.

The Salutation Consortium is developing another service-discovery standard called Salutation [33]. The key component is the Salutation Manager (SLM), which acts as a broker between services and clients, much like the lookup service of Jini. Multiple SLMs can communicate reliably with each other using Transportation Managers (TMs) to exchange service registrations, and TMs understand different network transports to enable seamless integration. To facilitate interoperation between heterogeneous services and clients, SLM may work in the middle to define the message formats used by the communicating endpoints. Unlike Jini and UPnP, however, Salutation does not provide lease-based service access. Instead it allows a client to ask SLM to periodically check the status of desired service to see if they are still available. Salutation also provides a slim version of its architecture for devices with small footprints.

Service Location Protocol (SLP) is a standard from Internet Engineering Task Force (IETF) [20]. The main components include the User Agent (UA), Service Agent (SA), and Directory Agent (DA). On behalf of the services, SAs advertise their location URLs and descriptive attributes, while UAs discover this information for the clients. A DA acts as the central cache storing the registration information for the services on the network. Note that the service registration with a DA is lease based and must be renewed periodically. SAs and UAs can discover DAs either actively (with a multicast request) or passively (by listening for pe-

riodic DA announcements). They may also use DHCP, if appropriately configured to support SLP, to find local DAs.

ZeroConf, also called Rendezvous, is another IETF standard. Like UPnP, it can automatically configure devices' IP address [3]. It has two extensions to provide service-discovery functionalities: one to leverage SLP and the other to use Domain Name Systems (DNS). For the second approach, each service sends its DNS Resource Record to a known multicast address (224.0.0.251) on port 5253, and so does the query from a client. There is a small DNS responder running on each ZeroConf host of the local network, which processes these multicast DNS records and queries, to allow hosts and services to operate without the presence of a DNS server.

We now summarize these service-discovery standards. Jini is a Java-based architecture while other protocols do not enforce a single language implementation. UPnP uses XML-based service description while others use less-expressive attributes. On the other hand, SLP allows query operators other than equality tests. All protocols except Salutation have a concept of leasing, either for service registration or access, to improve system robustness against abrupt failures. All protocols except SLP support asynchronous events notifications to improve system reliability and scalability. None of these protocols require manual installation of service-specific driver on clients, and all the service brokers (such as the Jini's lookup service) can be automatically discovered by services and clients. The control points of UPnP and the directory agents of SLP are optional components, namely, the clients may directly discover services using multicast on a local network. Finally, UPnP does not explicitly support security while Jini has an optional encryption package. The only security mechanism of SLP is to authenticate the source of service registration. For further details of these protocols, both Helaland Richard have written excellent surveys [21, 31].

3 Mobility

Both the client and the service-provisioning devices may move in the environment. There are at least three issues caused by mobility. First, the network addresses of the devices may change, disrupting existing communications between services and clients. Second, the device may have to register with a new directory in the distributed middleware. We call this *handoff* between directories. Finally, device mobility may also cause the change of a service's name to reflect the location movement. In this section, we discuss the first two issues and leave the third to Section 5. For simplicity, here we only discuss the situation of service mobility. The handling of client mobility is similar.

Once a client has located a desired service through the discovery middleware, it may start to communicate with that service directly or through the middleware that acts as a message broker. If the client

communicates with the service directly, the endpoints must handle the change of network address by themselves. The middleware could send notification events to a client as its service moves and registers with the middleware for a new address. It is up to the client to track its service and send messages to the server's current address. One solution is to use mobile IP [29], but the client may still have to resend the messages undeliverable during the service's movement.

On the other hand, if the middleware acts as a message broker between the client and service, the service's mobility can be shielded from the client. A client simply sends its messages to the middleware and specifies the destination service. If the service is on the move, the middleware will buffer the message and deliver it as the service connects back. Intentional Naming System (INS) takes this approach to provide combined functionalities of service discovery and message delivery [1]. INS consists of a set of Intentional Name Resolver (INR), to which a service registers an attribute-based name. The service name propagates through the INR overlay so eventually every INR will have the name of all services. A client sends a message to an INR and specifies its destination as a name query. The message traverses through the overlay with the name query resolved hop-by-hop, eventually reaching the service whose name matches the query. INS supports both unicast and multicast (so all services whose name matches the query will receive that message). Similarly, Service-oriented Network Sockets (SoNS) allows a client to specify a destination by name query, which can be resolved by any service-discovery system [32]. The message is then delivered to the identified service using direct socket communication, and all these details are hidden from the application layer.

For scalability reasons (Section 4), a mobile middleware may contain a distributed set of directories, each responsible for the services in one region. As a service moves from one region into another, it may need to *handoff* from its previous registered directory to the one in the new region. The handoff process includes moving the registration information from the old directory to the new one and triggering a notification event about this movement to the client. Designed for discovering moving network services, the Mobiscope system dedicates one directory for each rectangular region of geographic space [17]. As the service moves across regions in the environment, its name registration will always be routed to the directory responsible for its current location. The amount of traffic caused by handoff is determined by the region granularity, the number of mobile services, and their mobility patterns. To cope with failures, Mobiscope does not explicitly remove the service registration from the old directory. Instead, it employs a soft-state approach similar to the leasing concept. Since the service has to periodically renew its registration, the old registration will eventually expire and be removed from the old directory. While causing more network traffic, this leasing approach is simpler to implement and reduces the chance of inconsistency due to the failures during handoff

process.

It is worthwhile to distinguish physical mobility and network mobility, which are two orthogonal issues. As a service moves physically, it may or may not change its network address. For many service-discovery protocols surveyed in Section 2, a service has not “moved” as long as it is still in the same subnet. Using the scenario in Section 1, the communication between the location service and Alice’s mobile phone will not be disrupted as long as the APs with which the phone associates are all in the same subnet. In other words, these discovery protocols define “region” as the network subnet. Other protocols, such as Mobiscope, define “region” as geometric rectangles that have nothing to do with network boundaries.

In summary, mobility may cause changes in the system. The mobile middleware may choose to only notify the clients about the changes and leave the handling to the clients. Or the middleware may choose to handle the mobility itself and hide the changes completely from the clients. The first approach increases the complexity of client development, while the second approach increases the complexity of the middleware system. A designer using the second approach, however, also needs to be careful about the assumptions it makes for the clients since it takes some control away from clients. For instance, if the middleware receives a message from a client while the specified service is on the move, how long should the message be buffered before declaring a failure or should the message be routed to another service newly available?

4 Scalability

If we consider tracking and monitoring the services that may move in a large area, scalability is an inevitable design issue for the naming and discovery middleware. For the simple scenario discussed in Section 1, thousands of mobile clients may query the middleware to find location services and the middleware may also need to support registrations from tens of thousands of service-provisioning devices, such as displays, printers, and sensors in the environment. The general approach is to partition the name space into several subspaces, each of which is handled by a directory. There are two issues to be addressed here: how to partition the name space and how to route the service registrations and client queries to the corresponding directory.

One natural approach is to divide the whole area into geometric regions and put a dedicated directory in each region, for services that have a “location”. Each directory is responsible for the name registrations of all the services in its own region. In the Globe system, directories form a hierarchy according to a region containment relationship [5]. Namely, a directory for building *A* is the child of the directory responding for the whole campus. If a query for printers in building *B* arrives at the directory of building *A*, the query is propagated upward to the campus directory and then pushed downwards to the directory of building *B*. Such

a hierarchical structure exploits the locality of registration and queries to partition the workload, assuming the queries from one region are mostly to find the services in the same region. The queries that do need to cross regions can be further reduced using extensive caching. DataSpace employs a similar approach where each three-dimensional *data cube* has its own directory [25]. The directories for the data cubes form a hierarchy as a big cube encompasses smaller cubes. Though also partitioning name space into directories using physical regions, MobiScope routes registrations and queries in a peer-to-peer fashion instead of using a hierarchical structure [17]. In Mobiscope, each directory maintains a *spatial routing table* that records all directories' regions. Given a query, Mobiscope directory examines its routing table to find all directories intersecting with the region specified in the query.

Instead of grouping services based on their current location, it is also possible to group them using other properties. For instance, INS/Twine and Camel divide the name space into a set of subspaces, each containing a single directory for the names in the corresponding subspace. Each name is then mapped onto one or more (for redundancy) of these subspaces using some kind of transformation. Both systems consist of a set of directories connected through a Distributed Hashtable (DHT) using peer-to-peer (P2P) protocols [7, 18]. A directory has a unique ID, and the DHT layer allows clients to send messages to a specified ID rather than an IP address. The message is then be routed to the directory whose ID is closest to the message's destination ID in the P2P network. Given a name registration specified as a set of attributes, the system hashes the name into several IDs to which the name will be sent for processing. A similar process occurs for query resolutions. A query is specified as another set of attributes, one of which is selected and hashed into a query ID to which the query is sent. Since a name is replicated in all the directories responsible for that name's hashed attributes, the query is guaranteed to be sent to a directory containing all the matching names. Here the matching criteria is that the name contains all the attributes specified in the query.

In summary, naming and discovery middleware needs to partition the name space into a set of distributed directories for large-scale processing of service registrations and client queries. Most of the existing service-discovery standards surveyed in Section 2 partition the name space based on network topology, but they do not provide a scalable mechanism to allow inter-directory queries. The service-discovery systems discussed in this section propose to partition the name space based on location or on a hash of other attribute values. The directories are organized into a hierarchical or a peer-to-peer structure for scalable registration and query routing. The directories cooperatively store the names and resolve the queries, reducing the workload on each individual. By increasing the number of directories, the naming and discovery middleware scales up to handle more services and clients. The tradeoff for the increased scalability generally is the increased request processing latency, since the registrations and queries need to be routed to corresponding directories

which may take several hops. For instance, in a DHT-based directory system a message may take $\log(N)$ hops to reach a destination directory, where N is the number of directories in the system.

5 Context Awareness

Mobile clients must discover and use services based on the current context. *Context* is the circumstance in which an application runs, and may include physical state (such as noise and light levels), computational state (such as network latency and bandwidth), and user state (such as location and current task). Consider the scenario discussed in Section 1, in which Alice finds nearby services (such as displays and printers) depending on her current location. Note that Alice's query is both a *persistent query*, which is continuously evaluated as the name space changes, and a *context-sensitive query*, which changes itself according to the context. On the other hand, the names for the display and printer may include their current location that should be automatically updated when the devices move (though not frequently). This requires *context-sensitive name* that change itself according to the context.

These scenarios place several requirements on the naming and discovery service. It must be flexible, so names can characterize the resource and so queries can express the desired characteristics; it must be scalable, to handle many names and queries; it must be fast, to support frequent name and query updates; and it must be responsive, to quickly notify applications about changes to the set of matches for their persistent query. Some of existing systems are designed to address these problems, such as the scalability as we discussed in Section 4. These scenarios, however, also place several requirements on the clients and services. Services must actively track their context so that they may update their name. Clients must also track their context so that they may update their persistent query. The context tracking and computation sometimes is not trivial and may well exceed the capability of mobile devices attached to a low-bandwidth network.

It is thus desirable to support both context-sensitive names and queries inside the infrastructure. A context-sensitive name registered by a service or a context-sensitive query requested by a client, specifies how it should be updated according to the context. The middleware is responsible then to track context data sources and perform context computation to appropriately update the names and queries. Every time the names or the queries are updated, the queries should also be re-evaluated to determine whether the answer to the query is also changed. By off-loading these duties from the services and clients, naming and discovery middleware improves performance and facilitates the development of both services and clients.

We have built a distributed system, called Solar, to support context-sensitive names and queries [11]. The core of Solar is a set of functionally equivalent nodes, called Planets, that connect with each other using a DHT-based peer-to-peer network [12]. Solar employs an attribute-based naming specification and each

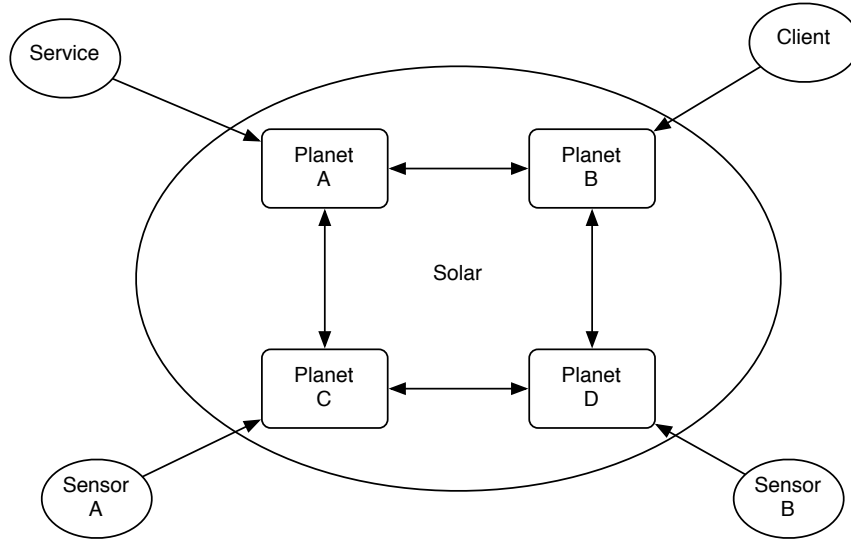


Figure 2: Architecture of the Solar system.

Planet hosts a name directory. We use an attribute-hashing scheme, similar to INS/Twine [7], to partition the name space onto all the Planets. We envision that context is computed by aggregating data from one or more physical or virtual *sensors*. These sensors, like other services, connect to a Planet and register a name.

Figure 2 shows the architecture of Solar. A service registers with Planet *A*, which forwards the service name to appropriate Planets based on attribute hashing. Now imagine that Alice’s phone sends a context-sensitive query to Planet *B* to find printers in the building:

```
$campus-locator = [sensor=locator, scope=campus];
$alice-locator = @filter("00022dd54817") <- $campus-locator;
query [service=printer, building=$alice-locator:building].
```

The context-sensitive query is defined in such a way that the value of some attribute is some context dynamically computed from other sensors. Here it first defines a campus-wide locator, “\$campus-locator”, that tracks all devices in the wireless network. Next it defines a locator that only tracks the location of Alice’s phone, by filtering through the output of campus locator using the phone’s MAC address. Finally the query asks for all the printers in the same building of Alice’s current location, where the value of “building” attribute equals to the one in the output of Alice’s locator.

Planet *B* handles the client’s context-sensitive query, and it needs to compute context from data of sensors that may connect to other Planets, such as on *C* and *D*. The planetary overlay thus also serves as an application-level routing layer for the sensor data [12]. The filter, as shown in the above, is called an *operator*. An operator is a data-processing component that takes one or more sensor data streams and outputs

another one. A context-sensitive name or query can specify a graph of operators to specify the logic for combining data from multiple sensors [11]. Solar is responsible to load and deploy these operators onto the Planets. Our experimental evaluation of Solar system demonstrated the advantages of middleware support for context-sensitive names and queries: reduced query latency and improved system scalability [11]. Later, we further evaluated the impact of context updates (change of the attribute values) on the system using a more realistic naming structure, and we showed that Solar’s resource discovery performed generally well in a typical dynamic environment [36].

Other systems also support limited context awareness in their service-discovery approaches. Instead of sending messages to a fixed address, both INS and SoNS allows clients to send messages to a destination controlled by a name query [1, 32]. The name query is always resolved with the current set of registered names, so the message may reach a different service at different times. To support context-sensitive names, however, services must monitor their context and update their name registrations. The iQueue system uses a non-procedural language for an application to specify data composition [13]. The data sources used for composition are determined by name queries. Like Solar, the iQueue system continuously re-evaluate the queries based on current context and rebind the data sources if necessary [14].

6 Security and Privacy

In this section, we discuss security requirements for a resource discovery system (like Solar [11]) that supports context-sensitive names. Figure 3 shows the information flow among a client, a resource, and a resource discovery system. Each resource, for example the camera, advertises its description as a set of attributes to the resource discovery system. The attributes, *building* and *room*, of the resource in the example in Figure 3 could be dynamic; that is, the values of those attributes change dynamically as the person carries the camera around. When a client issues a query that specifies conditions on attributes of a resource to the system in order to locate that resource, the system returns a list of references to the matched resources. For example, a query $?[building = sudikoff]$ matches the resources in the *sudikoff* building. The client obtains from the server the values of the other attributes of the resource as well as the reference to the resource. The client then chooses a resource (i.e., a camera) chosen from the list and accesses it to receive service. The client’s access is authorized by the resource if the client satisfies the resource’s authorization policies, which are usually defined by the resource owner.

We make a few assumptions on the resource discovery system in Figure 3. First, we assume that the attribute-value pairs of a resource are flat; that is, there is no hierarchical structure involved in the attributes. Second, we assume that the resource discovery system is administered by a single authority, although the

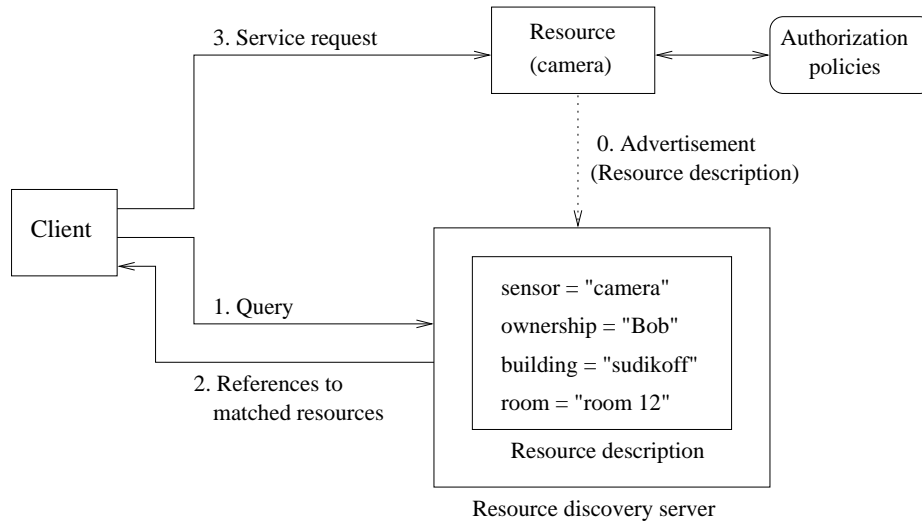


Figure 3: Information flow among a client, a resource, and a resource discovery system. The resource (camera) advertises its description with the four attribute-value pairs to the resource discovery system. The numbers in the labels of the solid arrows specify the sequence of the message flows.

system might consist of multiple servers in a distributed environment. Third, clients, resources, and the resource discovery system establish secure channels with a session key to communicate with each other while ensuring the secrecy and authenticity of the messages.

There are two kinds of information to be protected in Figure 3. One is the resource’s advertisement that contains the description of a resource. The other is the client’s query containing the conditions on a resource. The security requirements for the resource discovery system depend on whether the clients and resources trust the system not to disclose their confidential information. We first discuss the case with a trusted system. After that, we discuss the case with an untrusted system.

In the first case, we only need to consider the protection of resource descriptions from an unauthorized client, because the information in each client’s query is shared only with the system. There are several existing resource discovery systems [16, 30] that require a client issuing a query to have the privilege for accessing a resource to obtain the reference to that resource; that is, the client will not be aware of existence of the resources to which the client does not have access. Those systems assume that the authorization policies on each resource are static; the policies do not change dynamically. In those systems, the client avoids accessing a list of received resources sequentially until an accessible resource is found. There are, however, two additional requirements to protect the resource descriptions in pervasive computing.

First, the system must protect each attribute of a resource description with a different authorization policy, because an attribute of a resource description might contain confidential or private information. For example, suppose that Bob advertises his mobile camera with a wireless network interface as a resource to

the resource discovery system, and he describes the device with the attributes *ownership* and *location*. The *location* attribute contains the current location of the camera. A query that specifies the value of *location* attributes produces a list of resources including Bob's camera if the value in the query matches the current location of the camera. Bob may be willing to allow other people to access the video stream from the camera if his identity is not disclosed to them. However, if the value of the attribute *ownership* of the camera is also accessible to the client, the client is able to keep track of Bob's location, and Bob's privacy might be compromised. Because many people are concerned with location privacy [8, 19, 22, 28] in pervasive computing, the location attribute of a device should be protected appropriately to preserve the users' privacy.

Second, the resource discovery system must support context-sensitive authorization policies that consider the situation of the users and environments. For example, suppose that a patient in a hospital carries a mobile device that monitors his medical condition, and that the description of that device, which contains the attributes *ownership* and *location*, is maintained in the resource discovery system. The administrator of the system could define a context-sensitive policy that says that "a caregiver of the hospital is granted access to the attribute *location* of the patient's mobile device only if the patient's medical condition is critical." This authorization policy is necessary to protect the patient's privacy in a non-critical condition.

To make a context-sensitive authorization decision requires the resource discovery system to refer to context information (e.g., a patient's medical data). Several existing authorization systems [2, 4, 15, 28] that support context-sensitive policies take a centralized approach; that is, those systems have a central server that collects all the context information necessary to make authorization decisions. However, the centralized approach does not work if context information is maintained by different servers that do not trust the resource discovery system to protect their confidential information. Instead of collecting detailed context information, the resource discovery system could make the authorization decision without obtaining patient's medical data by collaborating with a medical server that maintains the patient's medical data and that tells the resource discovery system whether the patient's condition is critical. This collaboration is possible only if the resource discovery system trust the medical server to provide correct information. Minami et al. [27] generalize this idea and built a secure context-sensitive authorization system that does not require a centralized server trusted by every principal. The system performs an authorization query involving multiple principals in different multiple administrative domains. The proof tree for the query is decomposed into multiple subproof trees produced by different hosts so that each participating principal's confidentiality and integrity policies concerning the rules and facts contained in the proof tree are preserved.

We now analyze two existing systems that protect resource descriptions. Berkeley Secure Service Discovery Service (SSDS) [16] protects the description of each resource by checking whether the client is

granted access to that resource. In SSDS, the Capability manager converts an access-control list (ACL) that is used to protect a resource into a capability for accessing the corresponding resource description in the directory server. If a client is in the ACL, the client receives from the Capability manager a capability, which can only be used by that client. Because the clients maintain their privileges, the system does not have to maintain any authorization policies to protect the resource descriptions.

Raman [30] extended the intentional naming system (INS) [1] to route a client's message only to resources that the client has the privilege to access. The major difference from SSDS is that INS maintains the authorization policies of each resource as the hidden attribute of that resource. In INS, each Intentional Name Resolver (INR) traverses an INS name tree that encodes the hierarchical arrangement of the attribute-value pairs of resources to map a resource name (name specifier) to a destination node to which a client's message is routed. Each node in an INS name tree is associated with an ACL to reduce the search space in the tree. When the matching process in INR reaches a node whose ACL does not contain the client, it prunes the subtree under that node from the search space. INS associates each resource's original ACL with the corresponding leaf node in the INS name tree, and derives the ACLs for intermediate nodes by taking the union of the ACLs in its children nodes.

We now discuss the second case where the resource discovery system is not trusted by the clients and resources. The clients need to protect their queries and credentials (or capabilities) from the system, because queries might imply the clients' interests, which should be kept confidential and credentials might imply confidential relationships with the resources. Similarly, the resources need to protect their advertisements from the system. Because it is difficult to hide that information completely from the resource discovery system, some existing research projects focus on minimizing information to be disclosed to the system.

Splendor [40] is a directory service in pervasive computing, and its focus is to protect the privacy of resource owners from the directory server. In Splendor, each resource advertises its description to the directory service through a proxy server trusted by that service. Therefore, the directory service cannot associate attributes of the resource that contains confidential information with the identity of the resource owner. Splendor, however, does not address the issue of traffic analysis by an adversary who is capable of monitoring all the network traffic.

PrudentExposure [41] minimizes disclosure of a client's credentials to a directory service using the technique of hash summarization with a Bloom filter [9]. In PrudentExposure, each credential corresponds to a domain that contains multiple resources, and the credential allows a client to access all the resources in that domain. Therefore, a client wants to know which domains the directory service supports while minimizing the disclosure of his credentials. The client sends a bit vector of a fixed length to the service

instead of sending the actual credentials. The bit vector represents the client's credentials; that is, if a bit at a certain position in the vector is set, that means that the client possesses the corresponding credential. The position for each credential is decided by computing $mod(h(domain\ identity), n)$ where mod is the mod function, h is a hash function, n is the length of the bit vector, and *domain identity* is the identity of the domain specified in the credential. We assume that the bit vector is long enough to have no collision between two different credentials. Similarly, the directory service computes a bit vector that represent a set of domains it supports, using the same hash function. When the directory service receives the bit vector from the client, it intersects it with its own bit vector and returns it to the client. The returned vector represents the list of domains the client can access through the directory service. The service can figure out which credentials the client has only if the service supports the corresponding domain. The client is thus able to minimize the disclosure of his credentials. However, this scheme is not applicable to a resource discovery system that supports context-sensitive policies, because it is impossible to distribute static credentials to clients in advance.

In summary, resource descriptions maintained by a resource discovery system in pervasive computing contain confidential information to be protected appropriately. The major difference from traditional authorization systems is the necessity to support context-sensitive policies. The protection of a client's query and resource description from an untrusted directory service largely remains left for future research. The literature of trust negotiation [39, 38] may provide some insight here, because these protocols often focus on limiting exposure of information to the other party.

7 Summary and Future Work

Middleware support for naming and discovery is crucial for anytime and anywhere service access in a mobile and pervasive-computing environment. In this chapter we identify four challenging issues existing service-discovery standards do not address well: mobility, scalability, context-awareness, and security and privacy. While individual challenges are being addressed by the research community, it remains to see an integrated system addressing all these aspects. Our experience tells us that these issues interleaves with each other and a new system must have global design principles. For instance, scalability must be considered by many components, including overall architecture, name storage, message multicast, and security protocols. The end system should not become so complex that it is hard to diagnose and maintain, thus defeating the goal of naming and discovery middleware to facilitate discovery and interactions.

Acknowledgment

This research has been supported by NSF Award EIA-9802068, by DARPA contract F30602-98-2-0107, by DoD MURI contract F49620-97-1-03821, by Microsoft Research, by the Cisco Systems University Research Program, and by USENIX Scholars Program. This project was also supported under Award No. 2000-DT-CX-K001 from the Office for Domestic Preparedness, U.S. Department of Homeland Security. Points of view in this document are those of the author(s) and do not necessarily represent the official position of the U.S. Department of Homeland Security.

References

- [1] William Adjie-Winoto, Elliot Schwartz, Hari Balakrishnan, and Jeremy Lilley. [The design and implementation of an intentional naming system](#). In *Proceedings of the 17th ACM Symposium on Operating System Principles*, pages 186–201, Charleston, SC, December 1999.
- [2] Jalal Al-Muhtadi, Anand Ranganathan, Roy Campbell, and Dennis Mickunas. Cerberus: a context-aware security scheme for smart spaces. In *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications*, pages 489–496. IEEE Computer Society, March 2003.
- [3] [AutoConf](http://www.autoconf.org/). <http://www.autoconf.org/>.
- [4] Jean Bacon, Ken Moody, and Walt Yao. A model of OASIS role-based access control and its support for active security. *Proceedings of the sixth ACM Symposium on Access Control Models and Technologies*, 5(4):492–540, 2002.
- [5] A. Baggio, G. Ballintijn, M. van Steen, and A.S. Tanenbaum. [Efficient tracking of mobile objects in globe](#). *Computer Journal*, 44(5):340–353, 2001.
- [6] Paramvir Bahl and Venkata N. Padmanabhan. [RADAR: An In-Building RF-Based User Location and Tracking System](#). In *Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies*, Tel Aviv, Israel, March 2000.
- [7] Magdalena Balazinska, Hari Balakrishnan, and David Karger. [INS/Twine: A Scalable Peer-to-Peer Architecture for Intentional Resource Discovery](#). In *Proceedings of the First International Conference on Pervasive Computing*, Zurich, Switzerland, August 2002.
- [8] Alastair R. Beresford and Frank Stajano. [Location Privacy in Pervasive Computing](#). *IEEE Pervasive Computing*, 2(1):46–55, January-March 2003.

- [9] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.
- [10] Guanling Chen and David Kotz. [Context aggregation and dissemination in ubiquitous computing systems](#). In *Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications*, pages 105–114. IEEE Computer Society Press, June 2002.
- [11] Guanling Chen and David Kotz. [Context-Sensitive Resource Discovery](#). In *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications*, pages 243–252, Fort Worth, TX, March 2003.
- [12] Guanling Chen, Ming Li, and David Kotz. [Design and Implementation of a Large-Scale Context Fusion Network](#). In *Proceedings of the First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services*, pages 246–255, Boston, MA, August 2004.
- [13] Norman H. Cohen, Hui Lei, Paul Castro, John S. Davis II, and Apratim Purakayastha. [Composing pervasive data using iQL](#). In *Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications*, pages 94–104, Callicoon, New York, June 2002. IEEE Computer Society Press.
- [14] Norman H. Cohen, Apratim Purakayastha, Luke Wong, and Danny L. Yeh. [iQueue: A Pervasive Data Composition Framework](#). In *Proceedings of the Third International Conference on Mobile Data Management*, pages 146–153, Singapore, January 2002.
- [15] Michael J. Covington, Wende Long, Srividhya Srinivasan, Anind K. Dey, Mustaque Ahamad, and Gregory D. Abowd. Securing context-aware applications using environment roles. In *Proceedings of the Sixth ACM Symposium on Access Control Models and Technologies*, pages 10–20. ACM Press, 2001.
- [16] Steven E. Czerwinski, Ben Y. Zhao, Todd D. Hodes, Anthony D. Joseph, and Randy H. Katz. [An architecture for a secure service discovery service](#). In *Proceedings of the Fifth Annual International Conference on Mobile Computing and Networking*, pages 24–35, Seattle, WA, August 1999.
- [17] Matthew Denny, Michael J. Franklin, Paul Castro, and Apratim Purakayastha. [Mobiscope: A Scalable Spatial Discovery Service for Mobile Network Resources](#). In *Proceedings of the Fourth International Conference on Mobile Data Management*, pages 307–324, Australia, January 2003.
- [18] Jun Gao and Peter Steenkiste. [Design and evaluation of a distributed scalable content discovery system](#). *IEEE Journal on Selected Areas in Communications*, 22(1):54–66, January 2004.

- [19] Marco Gruteser and Dirk Grunwald. [Anonymous usage of location-based services through spatial and temporal cloaking](#). In *Proceedings of Mobisys 2003: The First International Conference on Mobile Systems, Applications, and Services*, San Francisco, CA, May 2003. USENIX Associations.
- [20] Erik Guttman. [Service Location Protocol: Automatic Discovery of IP Network Services](#). *IEEE Internet Computing*, 3(4):71–80, July 1999.
- [21] Sumi Helal. [Standards for Service Discovery and Delivery](#). *IEEE Pervasive Computing*, 1(3):95–100, July 2002.
- [22] Urs Hengartner and Peter Steenkiste. [Access control to information in pervasive computing environments](#). In *Proc. of 9th Workshop on Hot Topics in Operating Systems (HotOS IX)*, pages 157–162, May 2003.
- [23] Jeffrey Hightower and Gaetano Borriello. [Location systems for ubiquitous computing](#). *IEEE Computer*, 34(8):57–66, August 2001.
- [24] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. [System architecture directions for network sensors](#). In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 93–104. ACM Press, 2000.
- [25] Tomasz Imielinski and Samir Goel. [Dataspace - querying and monitoring deeply networked collections in physical space](#). *IEEE Personal Communications*, pages 4–9, October 2000.
- [26] David Kotz and Kobby Essien. [Analysis of a campus-wide wireless network](#). In *Proceedings of the Eighth Annual International Conference on Mobile Computing and Networking*, pages 107–118, Atlanta, GA, September 2002.
- [27] Kazuhiro Minami and David Kotz. [Secure context-sensitive authorization](#). *Journal of Pervasive and Mobile Computing*, 1(1), January 2005.
- [28] Ginger Myles, Adrian Friday, and Nigel Davies. [Preserving privacy in environments with location-based applications](#). *IEEE Pervasive Computing*, 2(1):56–64, January-March 2003.
- [29] Charles E. Perkins. [Mobile networking through mobile IP](#). *IEEE Internet Computing*, 2(1), January 1998.

- [30] Sanjay Raman, Dwaine Clarke, Matt Burnside, Srinivas Devadas, and Ronald Rivest. [Access-controlled resource discovery for pervasive networks](#). In *to appear in the Proceedings of the 18th Symposium on Applied Computing*, March 2003.
- [31] Golden G. Richard. [Service Advertisement and Discovery: Enabling Universal Device Cooperation](#). *IEEE Internet Computing*, 4(5):18–26, September 2000.
- [32] Umar Saif and Justin Mazzola Paluska. [Service-Oriented Network Sockets](#). In *Proceedings of the First International Conference on Mobile Systems, Applications, and Services*, San Francisco, CA, May 2003. USENIX Association.
- [33] [Solutation architecture](#). <http://www.salutation.org/>.
- [34] [Universal plug and play](#). <http://www.upnp.org/>.
- [35] Jim Waldo. [The Jini architecture for network-centric computing](#). *Communications of the ACM*, 42(7):76–82, July 1999.
- [36] Jue Wang. [Performance evaluation of a resource discovery service](#). Technical Report TR2004-513, Dartmouth College, August 2004.
- [37] Mark Weiser. [The computer for the 21st century](#). *Scientific American*, 265(3):66–75, January 1991.
- [38] William H. Winsborough and Ninghui Li. Protecting sensitive attributes in automated trust negotiation. In *WPES '02: Proceedings of the 2002 ACM workshop on Privacy in the Electronic Society*, pages 41–51, New York, NY, USA, 2002. ACM Press.
- [39] Ting Yu, Marianne Winslett, and Kent E. Seamons. Interoperable strategies in automated trust negotiation. In *CCS '01: Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 146–155, New York, NY, USA, 2001. ACM Press.
- [40] Feng Zhu, Matt Mutka, and Lionel Ni. [Splendor: A Secure, Private, and Location-Aware Service Discovery Protocol Supporting Mobile Services](#). In *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications*, pages 235–242, Fort Worth, TX, March 2003.
- [41] Feng Zhu, Matt Mutka, and Lionel Ni. [PrudentExposure: A Private and User-centric Service Discovery Protocol](#). In *Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications*, pages 329–338, Fort Worth, TX, March 2004.