Supporting Service Discovery, Querying and Interaction in Ubiquitous Computing Environments

Adrian Friday[1], Nigel Davies[1,2], Nat Wallbank[1], Elaine Catterall[1] and Stephen Pink[2]
[1] *Computing Department*
Lancaster University
Lancaster
LA1 4YR
U.K.
({`adrian,nat,elaine`}`@comp.lancs.ac.uk`*)*

[2] *Department of Computer Science*
University of Arizona
Tucson
Arizona 85721
USA
({`nigel,steve`}`@cs.arizona.edu`*)*

Abstract. In this paper, we contend that ubiquitous computing environments will be highly heterogeneous, service rich domains. Moreover, future applications will consequently be required to interact with multiple, specialised service location and interaction protocols simultaneously. We argue that existing service discovery techniques do not provide sufficient support to address the challenges of building applications targeted to these emerging environments.

This paper makes a number of contributions. Firstly, using a set of short ubiquitous computing scenarios we identify several key limitations of existing service discovery approaches that reduce their ability to support ubiquitous computing applications. Secondly, we present a detailed analysis of requirements for providing effective support in this domain. Thirdly, we provide the design of a simple extensible meta-service discovery architecture that uses database techniques to unify service discovery protocols, and address several of our key requirements. Lastly, we examine the lessons learnt through the development of a prototype implementation of our architecture.

Keywords: Distributed Systems, Mobile and Ubiquitous computing, Service Discovery, Service Interaction, Middleware

1. Introduction

With the advent of wireless networks and mobile devices, modern computer networks are becoming increasingly dynamic places. The need to simplify the administration and interconnection of networked devices has led to the development of a number of 'service discovery' protocols, such as the Service Location Protocol (SLP [13]), Home Audio/Video

Interoperability (HAVi [15]), Universal Plug and Play (UPnP [24]) and Sun's Jini Connection Technology [29].

Researchers have also demonstrated how similar techniques can be applied to aspects of mobile and ubiquitous computing [2],[16]. We believe that this trend is set to continue and that future service environments will be characterised by a heterogeneous mix of services and protocols. Crucially, we believe that devices, applications and users will need to interact with multiple, potentially specialised, service location and interaction technologies *simultaneously* in order to construct successful ubiquitous computing applications.

In section 2 we examine the four of the key contributions to provide background in the area of service discovery and interaction. We present a set of short illustrative scenarios in section 3, that help us identify some important ways in which current protocols could be enhanced to better support ubiquitous computing applications. In section 4, we build on these scenarios to present a more detailed analysis of what, in our opinion, are the main requirements for supporting such applications. Section 5 reports on our work toward developing an integrative meta-service discovery framework that utilises database techniques to provide a uniform API across multiple service discovery protocols. We demonstrate how in a prototype implementation of our architecture we have been able to bridge the UPnP and SLP protocols. We examine the merits of our approach and identify the lessons and transferrable concepts from developing our prototype. Lastly, in section 6, we present our concluding remarks.

2. Prevalent Service Discovery Protocols

The desire for zero-configuration networks in which devices can join the network and dynamically discover the services that they need has motivated the development of a number of service discovery and interaction protocols, including Jini [29], HAVi [15], SLP [13], UPnP [23], Salutation [31], Cooltown [21] and academic research initiatives [16]. We begin by briefly describing the key architectural features of four of the most commonly deployed protocols.

2.1. Universal Plug and Play

"Universal Plug and Play (UPnP) is an initiative to bring easy-to-use, flexible, standards-based connectivity to consumer networks, whether in the home, in a small business, or attached to the global Internet."[23]

From a technology standpoint, UPnP is a suite of protocols and system services that in concert provide service discovery and control in small to medium size IP networks.

Service Advertisement. In home networking installations, where centralised network management is undesirable, the developers of UPnP anticipate that AutoIP [32] will be used to assign IP addresses to UPnP devices. Once configured, devices periodically advertise themselves using the HTTP like service announcements (the Simple Service Discovery Protocol, SSDP [11].)

Discovery. SSDP clients interested in accessing services run a UPnP 'control point' which passively waits for service announcements or actively probes by sending a multicast search request (`M-SEARCH`). `M-SEARCH` forces devices and services matching the specified search criteria to respond with directed unicast service advertisements[1]. Control points are independent of each other and are not aware of the search requests made by other clients (we will return to this point in section 4.)

Interaction. Once the client has obtained a URL for the service from the SSDP advertisement, it retrieves an XML 'description' of the device and the services it offers. A device may offer multiple services; for example, a video recorder might contain a timer service in addition to its recording and playback services. The service description includes a URL to an XML description of the state variables and methods associated with the service and a URL to the 'presentation web page' that allows human interaction with the device.

Attributes. UPnP devices have a set of associated state attributes which reflect the underlying state of the device. Clients can invoke operations using the Simple Object Access Protocol (SOAP) [27] to invoke services and adjust the service attributes. For example, a service representing a networked digital camera might have an attribute called 'auto-focus' with possible states 'on' and 'off'; clients that wish to change the camera's auto-focus mode would simply change the value of the auto-focus attribute.

Events. The architecture requires that when the state of a service changes it generates an event using the Generic Event Notification Architecture (GENA). Clients register for these events in order to ensure all clients maintain a consistent view (e.g. to ensure consistency between the camera's front panel and a remote application controlling the camera).

[1] A search request is sent during the initialisation of the in-built UPnP control point in the most recent Windows Operating Systems.

It is worth noting that UPnP compatibility is defined in terms of the on-the-wire protocol and is totally architecture, operating system and language agnostic.

2.2. Service Location Protocol

The Service Location Protocol (SLP) [13] provides the capability to discover, select and utilise network services in IP networks with little or no prior configuration. As identified in version 2 of the specification:

> "This is especially important as computers become more portable, and users less tolerant or able to fulfil the demands of network system administration."

In contrast with UPnP, the scalability of the protocol has been a primary design goal of SLP.

Discovery. In an SLP network, a client (or *user agent*) seeking some service, sends a multicast search request (`SrvRqst`). Any service agents that represent services matching the requirements specified in the `SrvRqst` respond with a unicast reply (`SrvRply`). The user agent makes use of a technique called 'convergence multicast' to discover all the services available on the network.

Convergence multicast requires that the client sends successive search requests (perhaps of increasing network scope (TTL)) and aggregates each set of responses it receives to form the final result set. Each successive request contains a list of previous responders, so that successive requests should only solicit replies from as yet unknown services.

Scalability. In larger networks, one or more *directory agents* can be instantiated to function as caching nodes. Service agents register with their local directory agent (`SrvReg/SrvAck`), the directory agent (if present) will respond to the user agents' requests on behalf of the service agent (obviating the need for the convergence multicast searches). Both the user and service agent must 'discover' the directory agent during initialisation. This process is achieved either actively using the normal service discovery process (a multicast `SrvRqst`), or passively when one of the directory agents periodically advertises itself.

Services may be placed into administratively assigned 'scopes'. A scope is an arbitrarily assigned string that groups a number of services into a collection to aid scalability. Clients may be assigned one or more scope identifiers that act as filters, limiting the client to detect only services within those scopes.

Naming. SLP services are classified into types defined by naming authorities (the default authority being IANA). A printer for example, might be represented by the URL '`service:printer:lpr://hostname`'; describing both the type of service, and in this instance, the protocol required for contacting the service. Service requests (`SrvRqst`) may be further specialised by the inclusion of a predicate (based on LDAPv3 search filters) describing the attributes that the set of matching services must comply to (attributes are arbitrary name value pairs assigned by the service's administrator).

Attributes. The attribute request (`AttrRqst`) search provides a mechanism for user agents to discover which attributes apply to a given service type[2].

Interaction. Interaction with a specific service (and thus handling of changes in internal state of the service), is assumed to be handled by the specific interaction protocol (`lpr` in the case of the printer example) and is not included as part of the SLP specification.

2.3. Jini

Jini [29] is a distributed systems platform developed by Sun that supports discovery of Java based services. Interaction with Jini services is achieved by the instantiation of a local `ServiceTemplate` object representing a given service (the interaction protocol between the service's representative and the entity it represents is not defined by Jini, though Java's RPC mechanism (*JavaRMI*) or *JavaSpaces* platform [28] is normally assumed.)

Discovery. In order for a client to be able to discover a service it must first find a nearby lookup service using the Jini *discovery* and *join* protocols. The client instantiates a 'multicast response' service then issues a search for available lookup services using the multicast request protocol. Available lookup servers will respond to the client's multicast using the unicast discovery protocol. This protocol may also be used by clients to contact a lookup service if its' Jini URL is known a priori. Lookup services beacon multicast announcements to clients periodically to advertise their availability.

Attributes. Services in Jini are represented by serialised objects encapsulated to form an 'entry' stored in the lookup service or JavaSpace. Each entry provides simple comparison functionality to allow the service to be matched against a search template. Attributes for the service and search template are simply specified as object member variables.

Events. Clients may register an interest with the lookup service to look for the 'change' events generated as new services register (or old services revoke their service offers). Services themselves may act as event sources to provide notifications of change of internal state via the Jini distributed event protocol (events are synchronously delivered to interested parties via RPCs). Jini provides support for transac-

[2] Support for this kind of search is only compulsory for directory agents.

tional grouping of operations and event notifications to allow fail safe semantics to be built into service interactions.

Scalability. Service objects are leased to clients using the distributed leasing specification [30]. Client's wishing to use a service they have discovered apply to the server for a lease. The lease (encapsulated by a lease object) specifies a duration in relative or absolute time for which the client may continue to use the service (although says nothing about the guarantee of service offered to the client). The distributed leasing scheme is one of the key attributes of Jini that is used to promote reduction of protocol traffic and hence enhance scalability (e.g. if a service does not renew its lease with the lookup service, the service is assumed to be no longer available).

2.4. Home Audio Video Interoperability

The Home Audio Video Interoperability (HAVi) architecture [15] is designed to facilitate the construction of multimedia applications in home AV networks. HAVi is based on a IEEE-1394 bearer network that provides guaranteed bandwidth reservation for multimedia streams and run-time reconfiguration capabilities.

Device classification. HAVi Devices are partitioned into four distinct classes according to the level to which they implement the HAVi framework, these are:

Full AV devices offer full support for HAVi and the ability to control other HAVi devices in the network.

Intermediate AV devices act as controllers for some types of HAVi device.

Basic AV devices are controllable via HAVi, but do not control other devices.

Legacy AV devices do not support HAVi, just the underlying IEEE-1394 network.

Non-IEEE-1394 devices must have proxies in order to join a HAVi network.

Discovery. In order to discover a service in the network, the client device must communicate with the HAVi 'Registry' processes — an instance of which runs on every full and intermediate AV device. This process is achieved using the HAVi 'messaging system' (as IEEE-1394 is not an IP network, HAVi provides it's own transport abstractions for inter-device synchronous and asynchronous messaging.)

Service Registry. The registry maintains a list of all local HAVi software components resident on its node. These components may include:

- Device control modules providing a software interface for controlling the physical properties and functionality of the HAVi device (this may include a number of separate functional component modules (FCMs), e.g. a TV device might have separate tuner and display components.) Java applications can be written by application developers and installed in the registry; these applications can be thought of as device control modules that control software state rather than an underlying physical device.

- A Stream manager responsible for the establishment of paths for delivery of isochronous data (point to point or point to multipoint) between HAVi devices.

- A Resource manager for negotiating the reservation of local resources, particularly with respect to multimedia streams. The resource manager provides the mechanism for resolving conflicts where resource reservations cannot be accommodated.

- The registry may also include references to 'Havlets'. Havlets are Java applications that execute on full AV devices and provide user interfaces for controlling HAVi devices. A Havlet may provide a customised Java user interface or contain a 'Data Driven Interaction' (DDI) specification that is used to generate a basic user interface automatically.

A 'web proxy' FCM has been defined to allow HAVi devices to retrieve information from Internet based sources. Recent work has demonstrated that other IP based service discovery protocols (e.g. UPnP and Jini) can be bridged onto a HAVi network [7], [14].

2.5. Summary

By examining the prevalent service discovery and interaction technologies it is evident that they share a number of transferable concepts. For example, Jini, SLP and the Secure Service Discovery Service [16] all use IP multicast for device discovery. In addition, most can use some form of directory service to reduce the overhead of service discovery in networks comprising many services. However, despite the apparent similarity of these protocols, each offers functionality targeted to specific application domains. Each technology has its own way of providing representations for devices and services; the scoping of service advertisements; interaction with devices and notification of events.

One of the reasons for such heterogeneity is that each technology has needs dedicated features to support the intended applications — SLP is highly scalable and intended to serve enterprise networks; UPnP targets home and small office environments; while HAVi focuses on interoperability in home AV networks and motivating the inclusion of multimedia streaming and resource reservation functionality.

An in-depth comparison of these architectures can be found in [3] and [22]. Given the divergence of purpose in service discovery protocols, it is reasonable to assume that in emerging ubiquitous computing environments many such protocols will be in use simultaneously. In our opinion, future applications will need to utilise services across multiple domains simultaneously — this is a key motivation for the development of the platform we describe in section 5.

3. Application Scenarios

In this section we present a series of simple ubiquitous computing scenarios that highlight some of the important shortcomings in current service discovery protocols when applied to ubiquitous computing applications. We elaborate on these issues in section 4.

3.1. Efficiently finding 'Stateful' Devices

> Bob is a security guard who is patrolling an empty office building late at night. Bob's new hand-held computer vibrates suddenly, drawing his attention to a proximity map which illustrates the status of all the door and window locks in a nearby corridor — the map shows him that the first door to the left has not been locked. The display has not shown that any movement has been detected, so he locks the door and continues on his rounds.

Consider developing the above application using current service discovery protocols. Each lock or movement sensor could be represented by a network service. To build the application efficiently, the programmer needs an programming interface that allows her to query the availability of these services based on both location (to determine proximity) and the value of specific state variables (the state of the locks or movement sensors.)

In current systems, this operation could be very heavyweight — the client would need to obtain references to all of the services of type 'lock' in the network (possibly scoped by geography if the system supports attributes or name/value pairs in service advertisements) and then poll each of these services to determine their current status. In mobile environments (e.g. on Bob's hand-held) power and network constraints are likely to be adversely effected by this extraneous and time consuming communication. Careful design of the discovery protocol to facilitate searches based on state information would dramatically reduce the search time and bandwidth requirements.

3.2. Discovering 'The Most Appropriate' Service

> Alice is attending a meeting in an unfamiliar office building. She remembers that she has forgotten to print an important document and needs to find a printer urgently. She takes out her laptop and starts her service browser. The browser returns a list of printers ordered by proximity. Thoughtfully, someone has annotated the description of the printer second from the top of her list as being the one just outside her meeting room. She selects that printer and prints out her document.

To implement the above scenario, most service discovery protocols provide facilities for simple annotation using attributes and scopes. While attributes could be used to provide useful information about the capabilities of the service, where it is located or to whom it belongs, they are typically statically assigned by whomever deployed the service.

It would be helpful to Alice if when she browses available services of type 'printer', she could view and search comments about the capabilities of nearby printers that have been added by other users or earlier visitors, e.g. "this printer jams frequently", "this is the nearest printer to visitors in meeting room C29", "this is the printer most frequently accessed from this location", and so on. Support for attributing meta-data concerning the usage history, access profile and user experiences of a given service is not supported by any of the current service discovery protocols. We believe that no mechanisms currently exist for automatically integrating, authoring and collating additional meta-data for service location protocols in a general cross-platform way.

3.3. Interacting With Heterogeneous Protocols

> When Alice returns home, she's tired from her meeting and just wants to sit down in front of a good film. She reaches for the remote control, scrolls through her catalogue of favourite films and presses 'play'. The blinds adjust themselves to remove the last rays of sun from the screen, the hifi adjusts to 'cinema mode' and the film starts playing from her collection.

In the final scenario, Alice is using her home 'remote control'. Yet, to realise this behaviour would require controlling a number of devices in concert, irrespective of the particular discovery and interaction protocols used to control each of the devices. The AV devices are likely to be controlled using HAVi and will expose HAVi APIs allowing fine grained control of the media stream; the television system may be using UPnP, which provides control and event support; the lighting and blinds might be activated via X.10, which only provides a very simple control interface.

So, even in such a simple scenario we require the remote control application to have bindings across several platforms; processing events, service announcements and requests from different disjoint protocols. Interworking between these protocols is difficult to achieve using current approaches — requiring bespoke application development. No common API is available that allows such interworking to occur transparently.

4. Analysis

Existing approaches to service location and device interaction provide a solid foundation for building distributed service based applications. However, by examining scenarios such as those presented above, we can see that as their usage becomes more prevalent, a number of practical limitations will become apparent. In the classic case, where a client locates and interacts with a small number of services, many of the existing service discovery and interaction technologies work effectively. However, where a large number of clients interact with many services, issues of scalability and selection of the most appropriate services becomes significant. We believe a crucial issue in terms of scalability, utility and performance, will be the ability for applications to reason about service selection and interaction on behalf of lightweight (potentially mobile) clients. In the following sections we consider these points in more depth.

4.1. Interoperability

Given the existence of multiple heterogeneous service discovery and interaction technologies, one of the key issues for application developers will be how to deal with the diversity of device representation. UPnP defines XML schemas to represent devices, SLP uses defined URL syntax [12], HAVi uses device control module software elements and Jini utilises serialised Java objects (placing stringent base requirements on client applications, especially for mobile devices). Moreover, UPnP's GENA, HAVi's Event Manager and the Jini distributed event model provide incompatible mechanisms for monitoring device state changes. SLP, does not provide any such mechanism.

If one is to create an application that is to interact with more than one of these service mechanisms simultaneously, the application developer will face significant challenges. In our opinion, a unified mechanism for locating, interacting with and representing services is needed.

4.2. Scalability

As the number of clients and services in an environment increases, so the burden due to dynamic service discovery and interaction escalates. The *Windows Me* UPnP client, for example, attempts to locate all the local network services at initialisation. If the multicast search yields n service advertisements then n TCP based HTTP interrogations follow to gather the XML service descriptions (the user would then initiate a further n interactions to get the presentation pages for each device). In the simple case of one root device with one service type we have found that, according to the UPnP specification, 12 packets are generated (4 packets each sent 3 times to avoid problems with packet loss) each refresh interval (normally 30 minutes).

As a client joins or roams into the network, it issues an `ssdp:all M-SEARCH` request (3 times for reliability). Every service must respond to each `M-SEARCH` by unicasting its service advertisement to the client (again 3 times per response). A single search yields a total of 36 response datagrams (if no packets are lost). As the number of clients and services grow, networks will be significantly impacted by UPnP service announcement traffic.

Scalability should clearly be a prime consideration in the design of service location protocols. However, we believe that even with efficient protocols, further savings in bandwidth and consequent improvements in scalability can be achieved by providing directory services that aggregate service advertisements and cache queries from clients. In many current systems, seeking a service biased toward the client; it is the responsibility of the client to enumerate through candidate services by discovering and then interrogating each service in turn if the state of the service is important in the selection process (as illustrated by the security guard scenario in section 3.1).

A second client, seeking the same service or entering the same domain (e.g. the second security guard following the first along the same corridor) will, using existing techniques, be required to conduct exactly the same search procedure as the first client, despite the fact that the procedure will be almost identical. A network based intermediate agency (similar to the Jini lookup service or SLP directory agent) could be augmented to cache the results or partial results from the previous client interaction, reducing consumption of network bandwidth and the time taken to locate services by exploiting the commonality between clients. This technique is already a well accepted approach to minimising traffic over low-bandwidth networks and has been employed in distributed systems such as Rover [20].

4.3. Location-Based Services

Location-based applications (e.g. GUIDE [5]) represent a significant class of mobile computing applications. However, in current systems, constructing location-based applications is complicated by two factors. Firstly, current service location protocols tend to offer simplistic scoping models (e.g. based on arbitrary nomenclatures and attributes) such as a network domain or administrative multicast based scoping (SLP). Identifying a service that applies to a real world location (a room, or building) not mapping onto such a topology is difficult.

Secondly, the location of a service in a network is not necessarily indicative of the field of that service's scope. For instance, a service that offers weather reports for the specific geographical location in which you are currently standing may be provided by a server physically located elsewhere — such a service has two apparent locations; a *physical* location relating to where the service is located and a second *logical* location that applies to the scope relating to the user's context. Researchers have begun to suggest solutions to this problem [19].

4.4. Time

Current service location protocols help clients establish which services are available at the current instant or 'in the present tense'. We believe the temporal element (history, patterns of use etc.) could improve usability. For instance, a user may wish to locate the printer that they used the last time they visited a given environment, or the one they use the most frequently. The trails and histories of device access are useful from both an application context and an administrative point of view. An administrator may be interested in which clients have accessed a particular service and at what time, or what services were or were not available during a particular time window.

Time may also be an important factor in the specification of actions and events. In current service interaction schemes, no generic mechanism exists for specifying the relative timing, ordering or relationship between events and actions. Clients wishing to cascade an action (e.g. turn the coffee maker on) from an event, such as an alarm signal, must remain connected to the network at *the time the event occurs* to be able to initiate the corresponding action. This is clearly a problem for portable or battery powered devices.

We believe that the temporal element to service location is also potentially valuable in ubiquitous computing applications in which the application context needs to 'live outside' interaction with one particular device or end-system (e.g. to facilitate enacting a task that has a lifetime longer than the use of just a single device.) As the user picks up and utilises different services and devices, their usage profile, possibly with respect to other contextual attributes, needs be maintained outside of a particular interaction and accessible from any device they use. While we do not contend that support for these histories is necessarily the role of the service discovery framework, such protocols should facilitate the development of these mechanisms.

4.5. State

The current state of a device or service is an important component of service location. A user seeking to print a document in a hurry is probably only interested in printers that have both paper and a short queue of pending print jobs. Existing service location architectures provide facilities for the expression of state related information and the notification of change of state events (e.g. UPnP device descriptions and GENA). Reasoning about *dynamic* state is not typically part of the service advertisement or discovery phases of the protocols. A client must enumerate through candidate services, utilising both time and network bandwidth, in order to identify the most appropriate service (illustrated in the scenario in section 3.1.) The ability to find services based on their state is desirable, especially where partial network connectivity or low bandwidth is involved.

4.6. Security, Authentication and Access Control

Security is crucial to the successful adoption of service location protocols — can applications and users trust the services they find, and can the services trust the clients they serve? Authentication and certification mechanisms are required to ensure a suitable level of trust. Encryption and key distribution protocols are required to protect the exchange of sensitive information (these issues are being explored in the Ninja Secure Service Discovery Service [6]). SLP is one of the few service frameworks that specifies the (optional) inclusion of authentication blocks with protocol messages, allowing the differentiation of valid from bogus services.

Control is another important notion. The owner of a device might expect to be able to discover and control it and, ideally, prevent malicious tampering. There are likely to be a rich set of rules governing access to a given service, for instance, the social rules that bind a family together. A parent will wish to be able to control their television, but may also require that the system allows members of their family to control it, within certain parameters — a child might control the television while their mother is in the room, or in a way that the her parents approve of (e.g. not allowing a child to switch to an adult channel.) This level of social, role and contextual control is almost certainly outside the scope of a service location protocol, yet such protocols should provide the fundamental mechanisms to facilitate the construction of appropriate access control policies (access control lists, arbitration strategies and assignment of limitations to service parameters on a per client basis.)

4.7. Meta-data

The underlying model of most service location and interaction strategies reinforces the roles from existing distributed network infrastructures: service providers or administrators establish services within the network for clients to access. The implication of this 'administrative' role is that the person or application that deploys the service controls the service's description and thus the attributes by which it can be discovered and used.

If we consider the way people use services in everyday environments, we observe that people 'personalise' them, e.g. "I find this printer the fastest", "this projector belongs to my research group" and so on. These annotations are often subjective or role based, and may well change over time (e.g. a printer will always print on a certain size of paper, but may well move between rooms or be replaced by a faster model).

We believe that personalised, group and public meta-data will be important to the utility of service location and interaction protocols. We do not intend that everyone should be able to modify the service description of a device (one must respect the administrative boundaries and roles within organisations), rather we contend that service location platforms should provide hooks for linking into meta-information databases.

4.8. Multiple Device and Service Reasoning

Current service discovery paradigms provide mechanisms for locating specific services offering a particular type of functionality. Directory services (as identified in section 4.5) could support queries *across* service types, reducing the time and network bandwidth utilised to identify services. A user in an active building might issue a query equivalent to "All printers to which I am allowed access that is nearby has sufficient paper and is not behind a locked door" by correlating attributes from multiple services simultaneously (e.g. printer, door lock and location tracking services.) Current service discovery technologies do not allow for the construction of queries that are scoped by the consideration of services of different types, forcing more communication intensive resolution of such queries at the client application.

4.9. Wireless Access

In general there is a poor match between current platforms' network and end-system requirements and those found in mobile environments[3]. For example, many of the platforms use IP multicast for service announcements and this is likely to lead to significant levels of unwanted traffic between a mobile device and the fixed network. Moreover, many of the protocols used are extremely verbose (especially in the case of UPnP), further consuming network resources.

Finally, none of the existing systems offer any support for intermittent connectivity, especially in the case where services are *hosted by* mobile devices. If a mobile device offers a service to devices on the fixed network there is currently no mechanism for maintaining state information for that device when it is operating in disconnected mode. Moreover, event based mechanisms designed for keeping clients informed of internal state changes may not reach wireless or disconnected clients and therefore do not support strong consistency.

5. Design and Implementation of a Prototype Meta Service-Discovery System

The analysis presented in the preceding sections has highlighted a number of issues that affect the application of existing service discovery techniques to ubiquitous computing settings. In an effort to address these issues, we have developed a new system based on a common structured query language (SQL) API. Our system acts as an integrative layer on top of existing approaches that offers a uniform expressive API to applications[4].

In this section we report on a proof of concept prototype developed to investigate the efficacy of our approach. In our work we have made the following design assumptions:

1. Services are based on a fixed (or statically deployed) network such as Ethernet or IEEE 802.11.

[3] One notable exception is the work of Hodes et al. [16] that focuses on service access by lightweight mobile clients. Note that mobile clients are not peer entities in this architecture and do not provide their own services to other clients.

[4] Existing approaches based on device databases (e.g. COUGAR [4]) seek to replace existing service location and interaction platforms. We seek to integrate and extend these technologies to achieve our objectives.

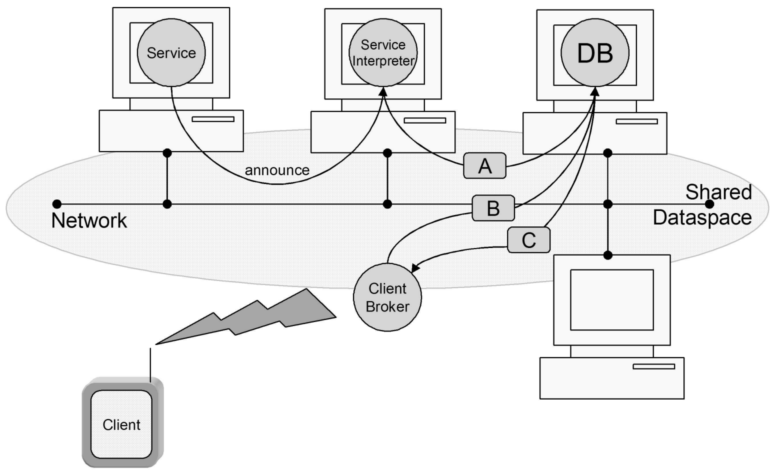

Figure 1. Design of the platform architecture. The service announces its presence on the network. A service specific interpreter converts the announcements into a tuple containing an INSERT query (*A*). A database component processes *A* and inserts the service into its database. To find a service the client emits SELECT query (*B*). All database components process B and emit matching services in a tuple (*C*). The client removes *C* and obtains a handle to the service.

2. The fixed network supports IP multicast.

3. Clients are relatively impoverished devices with communication limitations (e.g. power constraints) and wish to minimise protocol interactions.

We discuss our implementation in more depth in the following sections.

5.1. Design and Operation

The design and operation of our system is illustrated in Figure 1.

The system is composed of three main entities:

1. *Services.* Services utilise service location protocols (currently SLPv2 and UPnP) and these services are not modified in our approach.

2. *Service Interpreters.* Service interpreters watch for service announcements, convert them into SQL `INSERT` statements and emit them into a shared dataspace. There is typically one interpreter per protocol per subnet (more than one could be used for load balancing or robustness.) The UPnP interpreter subscribes to GENA events, converting changes of state into `UPDATE` messages.

3. *State Databases.* The state database removes messages containing SQL statements from the shared dataspace, executes the query on its database and returns one or more new entries as a response. Again, multiple database entities could be used to improve scalability and performance.

5.2. Intermediate Communication Layer

The system entities are linked together using a shared communication layer. In our current implementation we use a distributed tuple space platform called L^{2}imbo [8] for this communication. L^{2}imbo is an efficient, fully distributed tuple space implementation based on Scalable Reliable Multicast principles [9]. Although the use of a tuple space is not essential to our system, the use of a shared dataspace abstraction does offer us a number of desirable properties. Since producers and consumers are decoupled in both space and time [10], the system easily supports:

1. *Run time configuration and reconfiguration* — entities may be started, stopped, relocated and replicated transparently.

2. *Failure resilience* — the system copes well with failed or unavailable services (e.g. due to communications outages) as the queries persist in the dataspace until the service is restarted (or the request is handled by a redundant replica.)

3. *Low configuration overhead* — our platform entities are effectively self discovering.

4. *Evolution* — providing the interface to the dataspace remains constant, new services and applications may be introduced without affecting the existing entities (as state is explicitly passed via or intermediated by the dataspace.) For example, we could introduce and `lpr` printer monitor service to augment SLP with dynamic attributes without modifying other entities in our system.

Note that, although L^{2}imbo uses a number of techniques to reduce network load, the platform does generate traffic *in addition* to that already present from the service discovery protocols. However, since the state of the services is maintained across and between interactions, we believe the overall burden on the network *is* reduced, as services no longer have to respond directly to client search requests. We would not, however, suggest that this is the optimal solution for situations where services are located on low-bandwidth networks. Examining the performance of our system is a subject for future work.

L^{2}imbo is based on SRM-like distributed caches that reach eventual consistency by continuously monitoring a set of underlying IP multicast groups. Caches use a 'repair' mechanism to service missed tuples. For low power (e.g. hand-held or embedded) devices, we separate the platform into two components: the platform API, which runs on the mobile device, and the dataspace manager, which may run anywhere on the network (typically the mobile's point of presence.) The API communicates with the manager using UDP; so only API requests, rather than full cache maintenance traffic, need be propagated over the wireless link.

L^{2}imbo provides limited support for peer bridging and filtering between dataspaces [33], allowing federation into sub-domains. Additional work is required to improve the performance and bandwidth efficiently of this facility.

5.3. Heterogeneous Test Environment

As part of the development programme we have deployed a limited service test-bed in our office environment. The test-bed includes a mix of traditional networked devices, off the shelf components [17], [1] and novel networked services[5] and appliances ([25]).

Protocol heterogeneity is a key aspect of our test-bed. Our current services are based on the Microsoft and Intel UPnP toolkits and the OpenSLP [18] reference implementation.

5.4. Qualitative Results

In this section we discuss the lessons we have learnt by building our test-bed and evaluation system. Our implementation work has explicitly addressed the following issues:

Interoperability. We have demonstrated that clients can discover and access services across both UPnP and SLP protocols.

Programming Interface. SQL has proved to be a highly tractable common programming interface for locating networked services, with rich query semantics.

Scalability. Our approach is fully decentralised and highly reconfigurable, offering good prospects for scalability. Work is required to

[5] Including a Dallas Semiconductor 1-wire bus weather station based on the TMEX 1-wire drivers (supplied by Dallas Semiconductor), our own driver software and a UPnP wrapper developed using the Microsoft UPnP toolkit.

optimise the interactions between the dataspace manager and the client end-system.

State. SQL offers the ability to query for services based on any service types or attributes. In future versions we may be able to offer compensate for protocols that do not offer dynamic attributes (e.g. SLPv2) by developing service monitors that use device specific interaction protocols (such as LPR in the case of a printer). Our dataspace intermediator would make this process transparent to the end-user.

Meta-data. Our service state table can be updated by any tuple space client, allowing the addition of meta-data using simple 'JOIN' operations on the service database. The introduction of personalised meta-data is not currently supported in our existing implementation.

Multiple Device and Service Reasoning. Querying across services is the default behaviour in our system and has been shown to work effectively. We have yet to address how service protocol specific mappings occurs (e.g. how to reconcile attributes with different names or behaviours such as UPnP's DoubleSidePrint reconciling with SLP's DoubleSide.)

Wireless Access. This issue is only partially addressed in our implementation. By placing the platform API on the mobile client, we are able to minimise interactions with the service discovery protocols and reduce the requirement to use the network. However, we do not yet efficiently support the offering of services by mobile end-systems.

Our implementation work has yet to address the issues of time, security and heterogeneous interaction, that we identified in section 4. We believe we may be able to address some of the temporal issues by using temporal extensions to the query language, such as TSQL2 [26]. Client software is currently required to support all the interaction protocols for services they wish to interact with. Control, device interaction and transaction support using our paradigm is one of the topics for future development.

One area we are particularly keen to investigate is the concept of a distributed service transaction. Existing service location protocols do not provide any support for regulating access (c.f. locking), checkpointing of service state or rollback behaviour. In the future many services will correspond to physical devices with tangible as well as networked 'virtual' user interfaces; consequently we suspect that a strong transaction may be impossible to achieve. However, the notion of being able to control sets of devices in an atomic and predictable fashion is certainly alluring and we aim to see what is possible using existing service location and interaction paradigms.

6. Concluding Remarks

In this paper we have examined the four most commonly deployed service discovery protocols and, together with a set of illustrative scenarios, identified a number of significant areas in which we believe such protocols can be improved to better support ubiquitous computing applications. We presented the design and implementation of a prototype meta-service discovery architecture that uses a set of protocol specific interpreters together with database techniques to provide a unified interface across service discovery protocols; addressing some of the key vulnerabilities we have identified. We have observed that SQL does offer us a strong starting point for providing a succinct and efficient cross-paradigm API.

References

1. Axis Communications: 2002, 'AXIS Network Camera'.
2. Bahl, P. and V. Padmanabhan: 2000, 'RADAR: An In-Building RF-Based User Location and Tracking System'. In: *Proceedings of IEEE INFOCOM 2000*, Vol. 2. pp. 775–784.
3. Bettstetter, C. and C. Renner: 2000, 'A Comparison of Service Discovery Protocols and Implementation of the Service Location Protocol'. In: *Proceedings of EUNICE 2000*.
4. Bonnet, P., J. Gehrke, and P. Seshardri: 2000, 'Querying the Physical World'. *IEEE Personal Communications, Special Issue on Smart Spaces and Environments*.
5. Cheverst, K., N. Davies, K. Mitchell, A. Friday, and C. Efstratiou: 2000, 'Experiences of Developing and Deploying a Context-aware Tourist Guide: The GUIDE Project'. In: *Proceedings of MobiCom 2000*. Boston, U.S., pp. 20–31.
6. Czerwinski, S., B. Zhao, T. Hodes, A. Joseph, and R. Katz: 1999, 'An Architecture for a Secure Service Discovery Service'. In: *Proceedings of ACM Mobicom 1999*. Seattle, Washington, U.S.
7. Dara-Abrams, A. and K. Hofrichter: 1999, 'HAVi-UPnP Brdiging from the HAVi Perspective – Principals, Approach and Arhitectural Sketch – draft version 1.0'. Technical report, Sony Research Labs.
8. Davies, N., S. Wade, A. Friday, and G. Blair: 1998, 'L^{2}imbo: a tuple space based platform for adaptive mobile applications'. *ACM Mobile Networks and Applications (MONET): Special Issue on Protocols and Software Paradigms of Mobile Networks* **3**(2), 143–156.

9. Floyd, S., V. Jacobson, S. McCanne, C. Liu, and L. Zhang: 1995, 'A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing'. In: *ACM SIGCOMM*. Cambridge, MA, pp. 342–356.
10. Gelernter, D.: 1985, 'Generative Communication in Linda'. *ACM Transactions on Programming Languages and Systems* **7**(1), 80–112.
11. Goland, Y., T. Cai, P. Leach, Y. Gu, and S. Albright: 1999, 'Simple Service Discovery Protocol, Version 1.0.3'. IETF Internet-Draft. http://www.ietf.org/internet-drafts/draft-cai-ssdp-v1-03.txt.
12. Guttman, E., C. Perkins, and J. Kempf: 1999, 'Service Templates and Service Schemes'.
13. Guttman, E., C. Perkins, J. Veizades, and M. Day: 1990, 'Service Location Protocol, version 2'.
14. Harpe, H.: 1999, 'Phillips, Sony, Sun collaborate to bridge HAVi and Jini network architectures'.
15. HAVi Consortium: 2000, 'HAVi Specification, version 1.0'.
16. Hodes, T., R. Katz, E. Servan-Schreiber, and L. Rowe: 1997, 'Composable Ad-Hoc Mobile Services for Universal Interaction'. In: *Proceedings of ACM Mobicom'97*. Budapest, Hungary, pp. 1–12.
17. IETF, 'IETF Zero Configuration Networking (zeroconf) Working Group'.
18. IETF: 2002, 'OpenSLP 1.0.9a 15th May 2002'.
19. Jos, R. and N. Davies: 1999, 'Scalable and Flexible Location-Based Services for Ubiquitous Information Access'. In: *Proceedings of 1st International Symposium on Handheld and Ubiquitous Computing, HUC'99*. Karlsruhe, Germany.
20. Joseph, A., A. de Lespinasse, J. Tauber, D. Gifford, and M. Kaashoek: 1995, 'Rover: A Toolkit for Mobile Information Access'. In: *15th ACM Symposium on Operating Systems Principles*. pp. 156–171.
21. Kindberg, T., J. Barton, J. Morgan, G. Becker, D. Caswell, P. Debaty, G. Gopal, M. Frid, V. Krishnan, H. Morris, J. Schettino, B. serra, and M. Spasojevic: 2000, 'People, Places, Things: Web Presence for the Real World'. In: *Proceedings proceedings 3rd IEEE Workshop on Mobile Computing Systems and Applications, WMCSA 2000*. Monterey, California, U.S., pp. 19–30.
22. McGrath, R.: 2000, 'Discovery and Its Discontents: Discovery Protocols for Ubiquitous Computing'. Technical Report UIUCDCS-R-99-2132. http://choices.cs.uiuc.edu/2k/.
23. Microsoft Corporation: 1999, 'Universal Plug and Play: Background'.
24. Microsoft Corporation: 2000, 'Universal Plug and Play Device Architecture Reference Specification, Version 1.0'.
25. Schmidt, A., M. Strohbach, K. van Laerhoven, A. Friday, and H.-W. Gellersen: 2002, 'Context Aquisition using Load Sensing'. In: *Proceedings of the 4th Annual ACM/IEEE Internation Conference on Ubiquitous Computing (Ubicomp 2002)*. Göteburg, Sweden, pp. 333–350.
26. Snodgrass, R.: 1995, *The TSQL2 Temporal Query Language*. Kluwer Academic Publishers.
27. SOAP: 2000, 'Simple Object Access Protocol (SOAP 1.1), 8th May 2000'. *http://www.w3.org/TR/SOAP*.
28. Sun Microsystems Inc.: 1999a, 'The JavaSpaces Specification'. White paper, Sun Microsystems Inc.
29. Sun Microsystems Inc.: 1999b, 'Jini Architectural Overview'. White paper, Sun Microsystems Inc.

30. Sun Microsystems Inc.: 1999c, 'The The Jini Distributed Leasing Specification'. White paper, Sun Microsystems Inc.
31. The Salutation Consortium: 1999, 'Salutation Architecture Specification (Part 1), Version 2.0c'.
32. Troll, R.: 1999, 'Automatically Choosing an IP Address in an Ad-Hoc Ipv4 Network'. IETF Internet-Draft. http://www.ietf.org/internet-drafts/draft-ietf-dhc-ipv4-autoconfig-04.txt.
33. Wade, S.: 1999, 'An Investigation into the use of the Tuple Space Paradigm in Mobile Computing Environments'. Ph.D. thesis, Lancaster University, U.K., Computing Department, Faculty of Applied Sciences, Lancaster University, Bailrigg, Lancaster, LA1 4YR, U.K.