Composable Ad-hoc Mobile Services for Universal Interaction

Todd D. Hodes, Randy H. Katz, Edouard Servan-Schreiber, Lawrence Rowe
Computer Science Division
University of California at Berkeley
(hodes,randy,edss,larry)@cs.berkeley.edu

August 2, 1997

Abstract

This paper introduces the notion of "universal interaction," allowing a device to adapt its functionality to exploit services it discovers as it moves into a new environment.

Users wish to invoke services — such as controlling the lights, printing locally, or reconfiguring the location of DNS servers — from their mobile devices. But *a priori* standardization of interfaces and methods for service invocation is infeasible. Thus, the challenge is to develop a new service architecture that supports heterogeneity in client devices and controlled objects, and which makes minimal assumptions about standard interfaces and control protocols.

There are five components to a comprehensive solution to this problem: 1) allowing device mobility, 2) augmenting controllable objects to make them network-accessible, 3) building an underlying discovery architecture, 4) mapping between exported object interfaces and client device controls, and 5) building complex behaviors from underlying composable objects.

We motivate the need for these components by using an example scenario to derive the design requirements for our mobile services architecture. We then present a prototype implementation of elements of the architecture and some example services using it, including controls to audio/visual equipment, extensible mapping, server autoconfiguration, location tracking, and local printer access.

1 Introduction

Researchers have predicted that wireless access coupled with user mobility will soon be the norm rather than the exception, allowing users to roam in a wide variety of geographically distributed environments with seamless connectivity [37].

This *ubiquitous computing* environment is characterized by a number of challenges, each illustrating the need for adaptation: continuously available but varying network connectivity, with high handoff rates exacerbated by the demands of spectrum reuse; variability in end clients, making it necessary to push computation into the local infrastructure; and variability in available services as the environment changes around the client.

This paper investigates novel uses of a ubiquitous network, focusing on variable network services in the face of changing connectivity and heterogeneous devices. We propose that providing an "IP dial-tone" isn't enough; we must add additional service infrastructure to augment basic IP connectivity. Specifically, we describe an architecture for *adaptive* network services allowing users and their devices to control their environment.

The challenge in the design is developing an *open* service architecture that allows heterogeneous client devices to discover what they can do in a new environment, and yet which makes minimal assumptions about standard interfaces and control protocols.

The key elements of the architecture we have developed include: 1) augmented mobility beacons providing location information and security features, 2) an interface definition language allowing exported object interfaces to be mapped to client device control interfaces, and 3) client interfaces that maintain a layer of indirection, allowing elements to be remapped as server locations change and object interactions to be composed into complex behaviors.

Additionally, we have designed, implemented, and deployed in our Computer Science building the following example services:

- untethered interaction with lights, video and slide projectors, a VCR, an audio receiver, and an A/V routing switcher from a wirelessly connected laptop computer;
- automatic "on-the-move" reconfiguration for use of local DNS, NTP, and SMTP servers; HTTP proxies; and RTP and multicast-to-unicast gateways;
- audited local printer access;
- interactive floor maps with a standardized interface for advertising object locations;
- tracking of users and other mobile objects with privacy control.

The testbed for our experiments [18] includes Intel-based laptop computers with access to a multi-tier overlay network including room-sized infrared cells (IBM IR), floor-sized wireless LAN cells (AT&T WaveLAN), and a wide-area RF packet radio network (Metricom Richocet). We also leverage facilities in a seminar room augmented with devices that can be accessed and controlled through a workstation. The physical components of the testbed are illustrated in Figure 1.

Our infrastructure builds on the substantial work in mobility support provided by the networking research community. The Mobile-IP working group of the IETF [24] has made great strides in the routing aspects of the problem. Overlay networking [31] has demonstrated the feasibility of seamless handoff between Internet service providers and interfaces. The developing Service Location Protocol

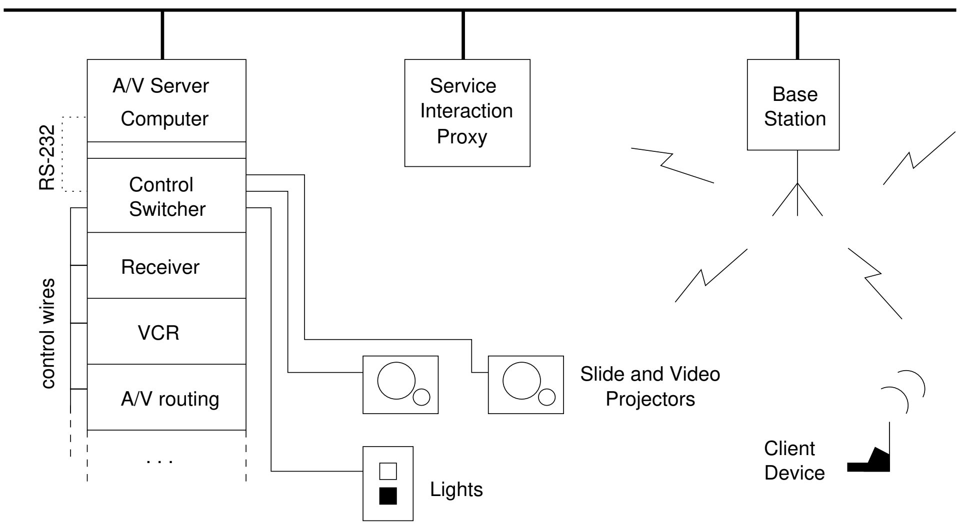

Figure 1: Project operating environment

[34] addresses resource discovery and management. Such efforts have been instrumental in motivating this work.

The rest of this paper is structured as follows. In Section 2, we discuss the key problem characteristics and provide a framework for a service provision architecture's core functionality. This is motivated by a scenario with a set of high-level functional requirements to be achieved. In Section 3, we detail our architecture's prototype implementation and the protocols that allow mobile clients to access the infrastructure. In Section 4, we describe the suite of example ad-hoc mobile services incorporated into it. In Section 5, we unify the design and implementation with some discussion of interrelationships, dependencies, and a layered view of the components. In Section 6, we discuss the relevant related work. In Section 7, we summarize future and continuing work, and finally in Section 8, we present our conclusions..

2 Designing a Service Interaction Architecture

We motivate our architecture for mobile services with the following scenario:

> You are on your way to give an invited lecture.
>
> After parking on campus, you take out your PDA with wireless connectivity, checking the list of local services available to you. You click on the map icon, and are presented with a campus-wide map that includes a rough indication of where you are. You select "Computer Science Division" from a list, and a building near you is highlighted. You walk toward it.
>
> As you enter the building, you glance down at your client device and the list of available services changes. Additionally, the campus map is replaced with the building floorplan.
>
> Using the new map, you find and enter the lecture hall. In preparation for your talk, you select "audio/visual equipment" and "lights" from the list of services, causing a user interface for each to appear. Your selections also cause the rooms' equipment to be located on the floorplan. You walk to the VCR and insert a tape.
>
> The lecture begins. As you finish the introduction, you dim the lights and start the VCR with taps on your PDA. At that moment, you realize you've forgotten your lecture notes. Using your personalized printer user interface, you retrieve a file from your home machine and instruct the closest printer to print the file. The printer's location appears on the floorplan.
>
> A minute later, you are notified the print job has completed, retrieve your printout, and return to finish the lecture as the videotape completes.

From this scenario, we can derive a set of required or desirable functions, presented in the next subsection.

2.1 Requirements Analysis

The problem is that users wish to invoke services — such as controlling the lights, printing locally, or reconfiguring the location of DNS servers — from their mobile devices. But it is difficult to obtain wide-spread agreement on "standard" interfaces and methods for such service invocation. The challenge is to develop an open service architecture that allows heterogeneous client devices to discover what they can do in a new environment, making minimal assumptions about standard interfaces and control protocols.

Implementing such a service architecture makes it possible to turn client devices into "universal interactors." An *interactor* is, broadly, a device that allows a user to interact with and modify his or her environment. Examples include electronic equipment remote controls and thermostats. A *universal interactor* is a device that adapts itself to control many devices — if it can discover their control interface. A universal interactor thus exploits what it finds in the environment, and varies its abilities as a function of location. It is not a particular hardware component, but instead a way of *using* an existing device.

Realizing such a capability requires (at least) five technical components: 1) device mobility, 2) network-accessible controllable objects, 3) an underlying discovery architecture, 4) mapping between exported object interfaces and client device control interfaces, and 5) composing complex behaviors from underlying primitive objects. These are now described in detail in the following subsections.

2.2 Device Mobility

A critical component of the scenario is device mobility. The client moves from a wide-area network to a local-area network, and between points in the local-area.

This functionality is available through Mobile-IP [24] and network overlays [17]. The former supplies IP-level transparency to changes in location, and the latter augments this functionality with a policy layer for managing connectivity to multiple available network interfaces and a mechanism for seamless (low-latency) hand-off. We build upon this network-layer functionality directly.

On top of this, we only require the ability to detect changes in connectivity with an event-delivery mechanism. Such a mechanism is required to implement automatic reconfiguration: when the client device discovers it has moved, it should check (or be notified) if a local instantiation of a remote service is available, and should autoconfigure to use the local service in this case. Concrete examples include DNS, NTP, and SMTP.

2.3 Controllable Objects

Most objects can be controlled. Doors and windows open; lights turn on; coffee-makers brew. Most physical objects provide only manual controls. A *controllable object,* on the other hand, is one that exposes the interface to which it responds to control requests or transmits status information. Additionally, it makes this interface accessible over a network.

To fit into our architecture, it is crucial that objects be augmented with an ability for network-based control. Open issues include addressability, naming, and aggregation of objects into a controllable unit. Individual controllable objects may be too numerous or the expense of individual control may be too high. For example, while it is possible to make every lightbulb its own controllable object, the sheer number of them in a typical building, the expense of assigning processing to each one, the difficulty of wiring each to the network, etc., would mitigate such a decision. Instead, control functionality could be assigned to a bank of lights, and what is augmented is the switch bank rather than all of the individual lightbulbs. In general, this means that the current infrastructure for naming – DNS – must be extended to include objects that do not have (or need) IP addresses. An alternative is to develop a separate infrastructure to match this need rather than overloading DNS. In the latter case, we can take advantage of the fact that instantiations of these name servers need only have a local, rather than global, scope.

Another approach for interacting with objects is to use video capture augmented with image processing ("computer vision") where applicable. Example uses of this approach include fine-grain object tracking, directionality sensing, and event triggers keyed to particular circumstances [22]. E.g., a camera can be used to detect the opening of a door or window. In this case, it is the camera that exports the control interface.

2.4 Resource Discovery

The function of a resource discovery protocol is to maintain dynamic repositories of service information and make this information available through scoped attribute queries. In contrast with DNS, the repositories' information is specifically local in nature.

We couch our discussion of resource discovery in the context of the Service Location Protocol [34], under development by the IETF Service Location working group. Although there are open issues in this domain, we avoid duplicating much of the relevant discussion here. Interested readers are pointed to the Internet draft and the Service Location working group mailing list.

From our local-area network perspective, the only mechanism we require is a function to allow mobiles to query the server for a mapping from strings to strings. We describe our own mechanisms for finding the correct local server and initializing the string mappings. Finding the a correct local server is similar to delivering the correct SCOPE attribute to the mobile host in SLP.

2.5 Transduction Protocols

A transduction protocol maps a discovered object interface to one that is expected by a given client device. It supports interoperability by adapting the client device's interface to match the controllable object's interface.

The issue with transduction protocols is how to map control functions into a UI supported by the portable device. As an example, assume a client device has a two-position switch widget for use with the local light controller. At a visited location, the light controller supports continuous dimming. In this case, the client may substitute a slider widget for the switch. If it cannot do this (or chooses not to), then the purpose of the transduction protocol is to map the on/off settings of the UI to one of the two extremes of the actual dimmer control.

Our solution is to transfer an entire GUI to the client in a language it understands, and when possible, augment the GUI with an interface description that starts with base data types and allows them to be extended hierarchically. A transducer that doesn't understand a level in the hierarchy can use elements below it. Alternatively, the interface description can be used directly to generate a rough GUI when no language implementation appropriate for the client is available.

The interface descriptions not only allow for data type transducers between client and server; they also provide a critical layer of indirection exactly where it is needed: underneath the user interface, allowing widgets to be transparently remapped to new servers in a new environment. This function is required to allow custom user interfaces for ad-hoc services, such as allowing a virtual "light switch" on the client device's control panel to always control the closest set of lights.

2.6 Complex Behaviors

Objects have individualized behaviors. We wish to couple and compose these individual behaviors to obtain more complex behaviors within the environment. For example, consider a scenario where music follows you as you move around a building. One behavior of the sound system is to route music to specific speakers. A behavior of location tracking services is to identify where specific objects are located, such as the user. A "complex" behavior allows us to compose these more primitive behaviors of sound routing and location tracking to obtain the desired effect of "following" music.

A key problem is that there is no common control interface for individual components. Furthermore, some behaviors may require maintenance of state that is independent of both subcomponents. An example of the latter is instructing the coffee maker to brew only the first time each morning that the office door opens. Another issue is the policy-level difficulty implied by this scenario: resolution of incompatible behaviors. If another user considers music to be noise, the visiting user's music may or may not be turned off in their presence, depending on seniority, social convention, explicit heuristics, or otherwise. At a minimum, the system must guarantee that it will detect such incompatibilities and notify the user(s) involved in order to avoid instability (e.g., music pulsing on and off as each individual behavior is interpreted).

Our solution is to use *interface discovery,* (i.e. any method through which new objects' input/output data types are learned) paired with the aforementioned data type transducers to allow objects to be cascaded much like UNIX pipes to achieve the desired complex behaviors. Additionally, we allow intermediate entities ("proxies") to maintain state that is independent of the constituent subcomponents. This allows for the incorporation of such features as conditional statements and timing information.

In our prototype, complex behaviors are written as scripts invoked by the delivery of particular events. These events are generated (when necessary) by the data type transducers that translate between the client user interface invocations and the RPC commands sent to a service daemon.[1]

3 Implementing Service Interaction

This section describes implementation details of the service interaction proxy (SIP), the service interaction client (SIC), and beaconing daemon (beacond) programs. These prototypes implement selected components of our overall mobile services architecture.

The prototypes allows a mobile host to enter a cell, bootstrap the local resource discovery server location, and acquire and display a list of available services. They also allows users to maintain a database of scripts to be executed when particular services are discovered for use in autoconfiguration, local state updates, and to trigger location-dependent actions.

If a user wishes to use a service it does not understand, the client first automatically searches its local cache for an interface to that service; if it is not there, the infrastructure is automatically notified and it attempts to send an interface description and GUI to the client.

3.1 Setup

A single copy of the "service interaction client" (SIC) program runs at each client device. Copies of the "server interaction proxy" (SIP) program run at domain-specific granularities. For example, a set of base stations in geographic proximity could be associated with a single SIP. Beaconing daemons (beacond) run at each base station.

An example SIC screenshot is shown in Figure 2. SIP and beacond use configuration files and command-line switches, and thus user interfaces are not shown.

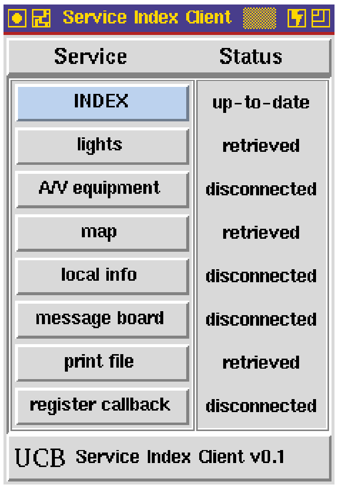

Figure 2: The SIC application GUI is currently a series of buttons that can be used to retrieve and invoke application interfaces.

Each SIP process maintains a database of the services and service elements that it provides to mobile hosts. An example startup file for such a database is listed in Figure 3. It contains three types of entries: SERVICES, VALUES, and PROPERTIES. VALUES are used for generic (key, value) lookups. These are useful for, e.g., detecting the need to update server addresses. SERVICES and PROPERTIES are used to specify what, where, and how services are available from that particular location. Each SERVICE has a unique name, and maintains PROPERTIES such as the version number, a pointer to an associated IDL file[2], pointers to particular language implementations of user interfaces for the service, and the geographic location (if any) for use with maps. VALUES and PROPERTIES may just be pointers to another SIP, allowing simple incremental deployment to subdomains and yielding a notion of topology.

3.2 Message-level Detail

The client enters a cell with a beaconing daemon. The daemon sends periodic broadcasts that contain the bootstrap address and port number of that cell's SIP. The client automatically registers with the base station to establish IP connectivity. It then requests the well-known meta-service INDEX, which returns a list of the services available. Based on the contents of the reply, the client renders labelled UI buttons for unknown services, remaps the location of running services, and executes scripts in a database to enable autoconnection

[1] Thus, even in the case where no translation is necessary, a null transducer must be interposed in order to allow detection of invocations. In other words, the transduction layer is the layer that provides the indirection.

[2] Use of the Interface Definition Language (IDL), a generic format for service interfaces similar in concept to a model-based UI, is described in Section 3.6

```
set NAME {
        Soda 405: High-Tech Seminar Room
}
set SERVICES {
        INDEX lights {A/V equipment} map printer {location tracking}
}
set VALUES {
        DNS {128.32.33.24 128.32.33.25}
        NTP {orodruin.cs.berkeley.edu barad-dur.cs.berkeley.edu}
        SMTP {mailspool.cs.berkeley.edu}
        ...
}
set PROPERTIES {
        lights {IDLfile ../helpers/lights.idl version 0.01 \
                location {132 210} appName-tk ../helpers/lights.tk \
                appArchive-tk ../helpers/405/lights405.tar.uue
                appName-tcl ../helpers/lights.tcl \
                appArchive-tcl ../helpers/405/lights405tcl.tar.uue}
        {A/V equipment} {IDLfile ../helpers/htsr.idl location {132 180} \
                version 0.01 appName-tk htsr.tcl \
                appArchive-tk "../helpers/405/HTSR.tar.uue"}
        ...
}
```

Figure 3: An abridged SIP services database example

and composed actions.[3] When a user requests a particular service, the client software checks its local cache of applications. If an interface supporting the requested application is not there, it asks the SIP for the service's "properties." This is a list of available interface descriptions and/or implementations. It also receives any service metadata (such as version numbers). It then chooses either to download a particular interface implementation (e.g., as a Java applet) or the generic interface description. The SIC then unpacks the received archives, transduces the interface description to match the device characteristics, and finally executes the GUI.

An example exchange of protocol messages for a client moving between SIP servers is illustrated in Figure 4.

3.3 Bootstrap

For a client to use services, it must first find the address of the local resource discovery server. In our architecture, this bootstrap above IP is minimal: there is an indirection embedded in the mobility beacons. This minimal bootstrap standardizes the interface for sending service advertisements without constraining the item to which it points. In general, it could point to any type of name server, thereby allowing variation in resource discovery protocols if this were desired.

3.4 Beaconing

Beaconing is required in a system to facilitate notification of mobility-based changes in the relative position of system components. Its use is motivated by inherent availability of physical-level hardware broadcast in many cellular wireless networks and the need to track mobiles to provide connectivity.

[3]The database currently resides on the client, but could additionally be retrieved from elsewhere by a proxy server to address client computational limitations.

Two issues arise once the decision to beacon has been made. The first is which direction to send them: uplink or downlink. The second is what information to put on the beacons, if any at all. (An empty beacon acts as a simple notification of the base station address, available in the packet header.) These are discussed in the following subsections.

3.4.1 Beaconing Direction

In terms of choosing whether to have client devices or infrastructure servers beacon, existing systems can be found which have made either choice. Client beaconing is used in both the Active Badge [13] and PARCTAB systems [26], while server beaconing was used in Columbia Mobile IP [14]. IETF Mobile IP utilizes both periodic advertisements *and* periodic solicitations.

One might expect the application-level framing [9] argument to hold here: different policies optimize for different applications' operating modes. This is indeed the case: there are trade-offs in such a decision, as it varies allowances for privacy, anonymity, particular protocols' performance, and scalability.

Specifically, some benefits of base station beaconing include:

- less power is consumed at the mobile by periodically listening than by periodically transmitting;
- finding a base station requires only a single message rather than a broadcast/response pair;
- mobiles need not transmit to detect when contact is lost;
- detection of multiple beacons can be used to assist handoff;
- anonymity of location is preserved for non-transmitting mobiles;
- allows possibility of "anonymous" access to some data known to the infrastructure (at a cost of management overhead and increased beacon size due to the piggybacking);

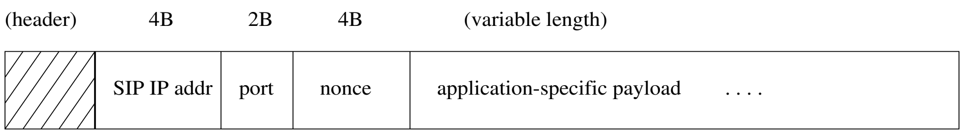

Figure 5: The service beacon encoding includes bits for the service interaction bootstrap and location queries. Not shown are Mobile-IP FA router advertisements or other potential application-specific piggybacked fields.

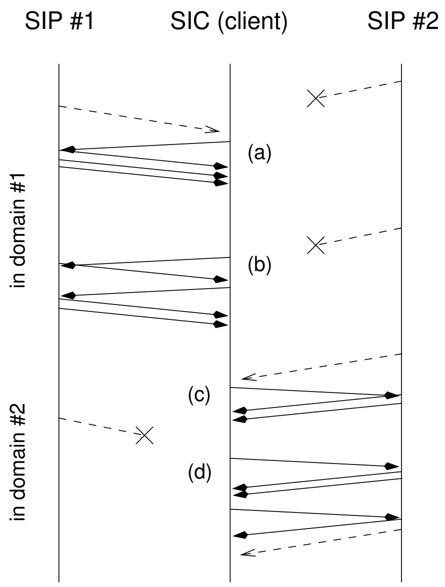

Figure 4: Protocol message timings for a client moving between SIP servers (dashed lines are beacons): (a) INDEX #1 request/reply (b) request/reply for “lights” IDL file and interface (c) INDEX #2 request/reply (d) “lights dim” button press retrieves new IDL file to remap RPC, then completes

- maintains a consistent mapping between geography and beacon broadcast cell;
- implies less beacon traffic per cell given the natural many-to-one mapping of mobile hosts to base station cells, assuming other parameters remain constant.

The benefits of having the mobile host beacon are complementary to the above and can be inferred from the list.

Our system uses base station beaconing. We believe this is the correct design choice for three key reasons: the support for user (rather than infrastructure) anonymity, better scalability in a low-bandwidth network where there are many MHs per BS, and because power is more precious on mobile devices. This choice aligns our design with other soft-state announce/listen protocols, such as the MBone session announcement protocol.

3.4.2 Beacon Augmentation

The second question is whether to augment mobility beacons with additional data. Doing so makes data available to mobiles *before* registration (in the Mobile IP sense). Possible uses for such piggybacked beacon data include:

- Mobile IP foreign agent advertisements;
- pricing information;
- advertisements ;
- time-variant data (e.g., NTP beacons);
- merging of periodic broadcasts to better amortize header and MAC overhead.

The utility of beacon payload augmentation is highly dependent on the direction of the beaconing, traffic patterns, and application mix. The argument against beacon augmentation it that orthogonal systems shouldn’t mix application data units that may have been “properly” sized by the application in a form of joint source-channel coding [23].

We choose to augment our beacons with bootstrap information, a nonce for scoping of services, and a dynamically configurable application-specific payload. The encoding is shown in figure 5. The nonce is discussed in Section 3.5; a usage of the application-specific payload is discussed in Section 4.3.

Merging Mobile IP router advertisements and the resource discovery protocol bootstrap may be a benefit, but allowing application-specific or other network-level fields is an area of active debate. We are still trying to quantitatively determine which data, if any, is best dedicated to these bits for optimizing reasonable client-driven workloads.

3.5 Security

Making services available to visitors brings up a host of general security issues, including those specific to the wireless domain [7, 11, 4]. In addition to standard cryptography-based security with passwords, capabilities (e.g., Kerberos), and public-key encryption, service interaction systems specifically require additional access control. This is due to our extension of devices to make them network-addressable entities. In general, global access control is necessary, but not enough; the expected behavior that environmental changes can only be affected by people in that environment (e.g., lights cannot be turned off by a person across the country) has been broken.

Maintaining this norm is important when extending existing human social metaphors into an environment with controllable objects.

We address this by embedding *nonces,* random fixed-length bit vectors, in the mobility beacons and requiring the current nonce to be included in all communications to servers.

Periodically changing the nonces in an unpredictable way and scoping the broadcast (implicitly via the cellular wireless network broadcast cell or explicitly with the multicast IP TTL field) prevents remote access from nodes *even on the access control list* that aren't local or haven't been separately multicast/unicast the nonce value. This pairs the geographic scoping of the environmental controls (what we cannot control) to the topological scope (what we can control). This nonce-based exclusion can be overridden, but by making the default access restricted, we better emulate the existing paradigm.

As for mobile code security, by transferring only a GUI to the client, it is probable that a sandboxed environment (such as Java, Safe-Tcl, or Janus [12]) can be used without constraining the service's functionality.

3.6 Client Interfaces

Clients can be computationally impoverished, have variations in display (color depth, resolution, screen size), support different interface paradigms (keyboard, pen, touch), and are often interchangeable with one another (and therefore not preconfigured).

Due to the need to support such end devices, especially extremely resource-poor PDAs, thin client interfaces must be available. They also must allow for different realizations on different hardware.

As part of our realization of automatic data type transduction, we have developed an initial, minimal grammar for such interfaces, which we call the *Interface Definition Language* (IDL). It is a paired-down version of a model-based UI [32], where UI elements are specified as a hierarchy of abstract types built on top of basic data types.

The IDL is used *in addition* to other reference language implementations of an interface, thus allowing for compatible but independent coexisting interfaces. Its main purpose is to expose the semantics of each services's control interface. Upon discovery of a service, the client device checks to see if a language implementation is available that it can support, and if not, uses the IDL file to learn the RPC calls and parameters that can be used to access the service. It additionally allows the device to adapt the representation to a format appropriate for the device's characteristics, and allows the user to place the elements manually and independent of one another for fine-grain control. Automatic layout heuristics can also be used.

Additionally, the use of IDL explicitly requires services to support a layer of indirection through the service discovery mechanism, allowing transparent remapping of individual interface elements. This indirection is critical for allowing services to be composed. As an example, the RPC command to spawn an audio conference can be rerouted to a client script that first turns down the volume of the room's music player and *then* passes along the original RPC.

Our current implementation has interfaces manually implemented in Tcl/Tk and an ad-hoc IDL specification tuned to Tk.

3.7 Prefetching

As an optimization, clients can prefetch the IDL files for active services. We illustrate with a concrete example from our prototype. As the user moves between rooms, the light controller application UI remains the same. When the user changes the lighting in a new cell, the client application sends the new SIP a request for the lights IDL file, enabling the RPC command invoked by the existing interface to be remapping so that the recipient will be the new server. This late-binding is used to conserve bandwidth on the wireless link; the total number of IDL files may be large and the client may use one only infrequently.

The problem with late-binding is that this entire operation latency seen by the end user;, and in practice it can be perceived as a possible error condition. (The button "doesn't work" for a number of seconds after it is invoked, and for this period it should probably be be grayed out in the UI.)

This delay can be minimized by transparently remapping the interface elements to the new server as soon as possible. To do so, we add one bit of per-service state, "active vs. inactive." This flag is set to "active" whenever there is an RPC call from that service, and reset to "inactive" by a timeout. Upon receipt of any beacons with a new SIP, services with the "active" bit set (and available in the new location) have their new IDL files prefetched automatically. (In our current implementation, the INDEX meta-service is always prefetched.)

Delays can be further minimized through mobility prediction [20], allowing prefetching in response to assumptions about user mobility patterns.

Prefetching is important in this domain because the delay experienced by an end user using a high-latency, low bandwidth wireless interface can be substantially less with prefetching and transparent remapping than with demand-paging.

4 Prototype Mobile Services

In addition to the prototype service discovery and interaction implementation, we have experimented with a number of services. These include maps that specify discovered objects' positions, autoconfiguration, location tracking with fine-grain privacy allowances, audited printer access, and interfaces to audio/visual equipment. We now describe each in turn.

4.1 Maps

Given a widely distributed service interaction system supporting very fine-grained services, management of even the subset of information available to the client becomes non-trivial.

We have experimented with using maps for explicit management of these services at multiple locations. Map content is separated into three domains: network connectivity (topology and link characteristics), physical geography (object locations and floorplans), and administrative domain (access rights, pricing, hierarchy).

Our prototype, of which an example view is shown in Figure 6, focuses on physical geography. Functionality includes the ability for objects to note their locations, and composability.

The location tracking support interfaces to the map using the same RPC calls that objects use to locate themselves.

The map protocol itself is a prototype based on using absolute positioning. It is designed to allow objects to position themselves without knowing exactly which map(s) the user is using. It is also designed to allow maps to maintain hierarchical ("containment") relationships in a distributed, extensible manner through the absolute positioning information.

The map protocol hierarchy is defined to be exactly the service interaction server hierarchy, and maintains the same characteristics

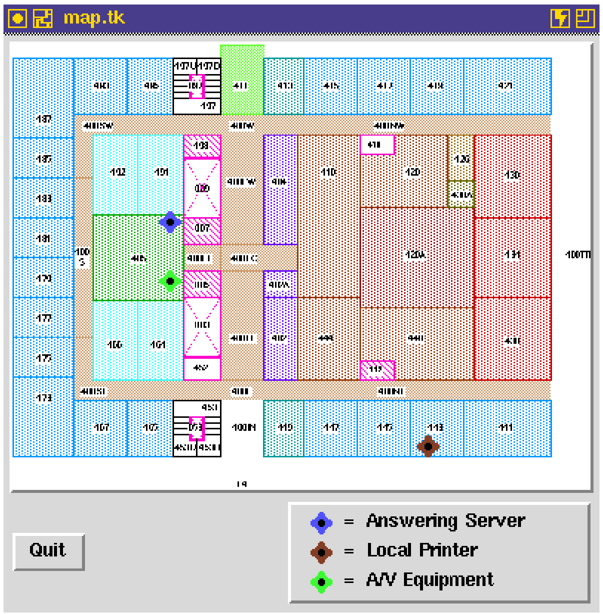

Figure 6: Map with discovered object locations, configured for a user in the RF cell including room 405. Clicking on entries spawns the interface to that entity.

that it is can be arbitrarily nested and extended without affecting other maps.

Each SIP database contains pointers to a map and to map metadata including the positioning information (latitude/longitude coordinates of two corners). This is similar to the proposed "LOC" DNS record type, extended to objects without IP addresses.

4.2 Proxy and Gateway Autoconfiguration

Proxy and gateway autoconfiguration is a lightweight service built atop the server autoconfiguration service. The difference between it and server autoconfiguration (cf., Section 2.2) is simply that proxies and gateways run in the infrastructure, between client and server. Thus, the infrastructure needs to explicitly support this in addition to the event delivery mechanism required from the device mobility support layer. Examples of such useful intermediate agents include web proxies that perform on-demand dynamic transcoding [10], network data caches, real-time media transcoders [2], and multicast-to-unicast gateways for multicast-unaware client devices (i.e., most PDAs).

Proxy, gateway, and server autoconfiguration is important in a mobile environment for more than just efficiency. Using the "best current practice" technique of hard-coding DNS servers as `/etc/resolv.conf` entries, if a user were to move from a location behind a firewall to one that is not, all lookups will fail until an out-of-band technique is used to find a new server and the entry is manually updated. The Network Time Protocol is dependent on server location due to its use of RTT estimation, and is therefore especially suitable for use with autoconfiguration. A failure to keep accurate time can break some security systems, notably Kerberos. Spawning a local RTP gateway transcoder for MBone sessions is necessary if movement has changed the bottleneck link between source and receiver [1].

Autoconfiguration also adds a level of fault tolerance. If a network link goes down, SIP beacons coming across the failed link will stop. The client will wait for other beacons to be obtained (c.f., overlay networking), and reconfiguration to the new servers will happen transparently.

Our current implementation simply allows the user to execute a script upon a change in a VALUE entry in the SIP database, which allows updates to HTTP proxy server addresses and allows for the spawning of an RTP gateway when necessary.

4.3 Location Tracking

Location tracking is addressed in other systems [28, 13], but they suffer from the limitation that the infrastructure must be trusted, and client devices must be turned off or not carried to ensure privacy. The latter especially defeats the purpose of enabling continuous access and universal interaction.

We use a novel technique to address this difficulty based on infrastructure beaconing and beacon augmentation: at bootstrap, the client is allocated a bit hopping sequence to use on the application-specific payload field of the beacons (conceptually similar to a frequency-hopping spread spectrum CDMA scheme). These bit sequences are then used to implement a low-overhead notification scheme that also provides anonymous location query replies with fine-grained policy control. It works by piggybacking the query (consisting of the address of the requestor) on the beacon, allowing the client to reply only if it desires.

The hopping sequence prevents snooping of updates, while piggybacking breaks the telltale asynchrony of queries. Together they allow selective exclusion of individual location requests (similar to telephony's "caller ID" feature) and protection from query snooping, thus maintaining privacy control at the client rather than requiring that other eavesdropping clients or proxy servers be trusted.

4.4 Printer Access

One of the most common examples in the resource discovery literature (and commonly requested end user service) is local printer access. In our implementation, after the discovery protocol finds the local printer and notes it on the map, clicking on it (or on the "print" SIC button) pops up a dialog box that can be used to send a client's postscript file to the local print server. The server then checks the data, logs the request, prints the file, and returns any status and/or error messages.

4.5 Room Interaction

The "high-tech seminar room," where weekly MBone broadcasts of the Berkeley Multimedia and Graphics Seminar take place, is equipped with a variety of equipment: two slide projectors, a light controller, a video projector, a VCR, a receiver, a DEC workstation, and an Intel PC. All the devices are attached to an AMX corporation control switcher. The DEC workstation talks to the AMX via a RS-232 serial connection, which allows the workstation to act as the control interface.

In 1993, Bukowski and Downs designed a library for accessing the AMX from a workstation for use in a similar room [8]. They also produced a client/server package utilizing the library. We leverage their work, along with a version of Tcl-DP [30] as the RPC interface, and extend it for use in our environment.

4.5.1 Design and Architecture

The application is built in a client/server framework supporting the principle of *application partitioning* [36]. Due to the potential lightweight nature of clients, the server is required to bear the brunt of the effort to support fault tolerance, access control, and other such duties. The current prototype implementation is minimalist, but features can be added to make the system far more robust with little or no change to the client-side code.

The server runs on our extended wish shell which includes the base AMX functions. The server opens an RPC socket and listens for requests to convert to AMX commands. It is also responsible for maintaining the hard state of the system. This leaves the clients free to act as only a UI and cache for soft state.

All communication with the AMX is done through an intuitive scripting language (e.g. `pushButton vproj power-on`). As an example, if the client executes a command such as pushButton receiver power, the routines of `AMXHelperLib.tcl` translate it into a remote procedure call for `send-AMX-command 1 3 75`. The request is forwarded through the wireless base station to the server application. The server, in turn, maps this request into a packet for the AMX, which it sends down the RS-232 serial link. Upon receipt of the message, the AMX's embedded controller routes the signal to its third slot (the one for the receiver) instructing it to to turn on channel 75 (power). This causes the correct stored IR frequency to be broadcast down a wire connected to the standard remote control IR port on the receiver. Other wires use mechanical calipers for manipulation of the slide projectors and variations in voltage levels to adjust the lights.

4.5.2 User Interfaces

Prior to our work with configurable device user interfaces, we implemented two separate monolithic Tcl/Tk programs for the room's control, one a superset of the other. The first handles only the lights, while the second handles the most useful buttons (showing them all is excessively complex). The latter interface is shown in Figure 7.

It was these implementations that led us to observe the utility of functional inclusion and the need for variability in the interfaces. It also led us to realize that independent objects an widgets should be independently addressable and composable.

With such a design, users could create unique UIs that makes the most sense for themselves by leveraging the scripting language and IDL or, eventually, by dragging-and-dropping individual elements to and from a toolbar.

4.5.3 State Management

Ideally, requests to the AMX could be idempotent, and no state would have to be maintained in the system. However, by the nature of the equipment to which it is attached, AMX requests are *not* idempotent, and cannot be coerced into idempotent versions. For example, if we want to turn on the receiver, the only request we can give is equivalent to "toggle receiver power," which may very well turn the receiver off. The only way to know the effect of this request beforehand is if long-lived state variables are maintained.

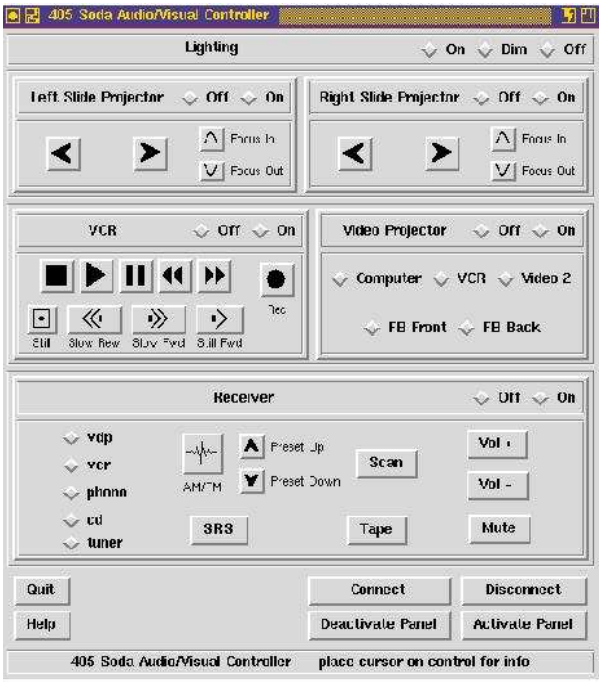

Figure 7: Screenshot of the monolithic user interface to the A/V equipment.

Because bandwidth between the client and server is valuable, state variable updates need to be minimized. Our server is responsible for maintaining consistent state between and during client sessions. Clients are responsible for querying the server for the state info upon connection initiation. For consistency, clients are required to send an update request to the server to ask for state changes. Clients may only commit the change upon receipt of an acknowledgment.

Another issue is dealing with inconsistencies due to manual events. Devices are unable to inform the AMX when a user presses a button on their front-panel. If a user wants to insert a video cassette into the VCR, he or she must first turn it on; the AMX does not register this manual event.

Since this is such a common problem, we accept that the state will at times be corrupted. We equip the user to correct such inconsistencies via the two buttons at the bottom of the client interface labelled "Deactivate Panel" and "Activate Panel." Whenever a discrepancy in the state occurs, the user can deactivate the panel. All the state-related buttons will then only modify the state variables (on both the client *and* the server) and *not* make requests to the AMX. This means the user can reconcile the information on the panel with reality, then reactivate the panel and once again right the system to a consistent state.

5 Discussion

Section 2 described the design issues and problem characteristics of service provision. Sections 3 and 4 gave details of a prototype implementation. We now attempt to better unify and situate these views.

5.1 A Layered View

A layered view of our architecture is presented in Figure 8. This representation exposes how the various mechanisms described in this paper interrelate. It also illustrates how alternative mechanisms could replace the particular ones we have chosen without affecting the overall service architecture. For example, if some to-be-determined Service Location Protocol SCOPE delivery mechanism were to replace our augmented beaconing mechanisms for location management, the interface discovery and data type transduction could remain unaffected.

The lowest layer is the *Beaconing* layer. It lies directly above the network and transport layers and implements the basic service bootstrap mechanisms. This includes the embedding of local server information in the beacon payload, the ability to implement security mechanisms through the indirection gained by the beacons, and the possibility for additional application-specific payload augmentation as a performance enhancement. Its most basic function is to *find* servers and users.

The *Query* layer uses server location information from the beaconing layer. It adds the ability to interact with found entities such as resource discovery servers and service interaction proxies. This functionality allows for the implementation of server, proxy, and gateway reconfiguration and scripting of complex behaviors. Its most basic function is to allow entities to *talk to* other entities.

The *Interface description* layer is built on whatever query protocol is exposed by the query protocol layer. It defines the set of possible interface descriptions and their semantics. Use of an interface description language enables device-independent interface downloading through the use of data type transducers and transduction protocols. Additionally, it supports the transparent remapping of particular interface elements to new servers as the mobile host moves. Its most basic functionality is to *map between* the client device interface and the interface advertised by discovered objects.

The highest layer is the *Application* layer. It uses the interface description language exposed by the interface description layer. Similarly to the OSI application layer, it encapsulates application-specific state or data not captured by the lower layers. Examples include attaching semantic meaning to particular names and defining relationships between data values.

Application Protocols
- maps with hierarchical inclusion and object interface
- A/V equipment with cannonical element names

Interface Descriptions — "map between"
- interface code downloading
- data type transducers
- transparent remapping of interface elements to new servers

Query Protocol — "talk to"
- server location reconfiguration
- scripting of complex behaviors

Beaconing — "find"
- location information
- security through indirection
- delivery of application-specific payloads and events

Network and Transport Layers

Figure 8: A layered view of the mobile services architecture.

6 Related Work

The Rover [15] and Wit [36] systems also recognized the need to split applications into a lightweight front-end and more heavyweight proxy at the last hop wireless link. Rover allows the pieces that comprise the partitioned whole to migrate between these two points, but the implemented prototype applications generally only exploit this for moving application data units such as mail messages, news articles, web pages, or calendar entries.

The Service Location Protocol (SLP) [34] is an example resource discovery and service registration mechanism that can also function as a fine-grained name service. We are interested in moving our resource discovery mechanism over to this evolving Internet standard. Open issues include its undefined "local" scope designation, an integral part of our scheme, and lack of an explicit scope hierarchy and peering equivalent to our use of pointers in a service database. Our mechanism for dynamically updating scopes could coexist with other mechanisms for similar goals in the proposal (i.e. having a SCOPE DHCP option).

The seminal PARCTAB [26] and Active Badge [13] systems, along with related work by Schilit [27, 28, 29], were among the first to attack the issues of client applications and network support for mobility in tandem. We borrow much from this work, including the focus on mapping, event notification, and support for impoverished devices. There are some key differences. We support distributed servers, rather than a centralized repository. We employ discovery mechanisms, interface code mobility, and generalize to heterogeneous devices; these are unnecessary in their local-area, homogeneous environment with pre-installed custom applications. We use server beaconing rather than client beaconing, and allow the beacons to bootstrap resource location, define scope, assist fault detection, and provide private location management.

A transportable X display [38] is a variation on interface code mobility; it moves users' *existing* interfaces as they move, not unknown applications' interfaces or interface descriptions. It has the advantage that applications need not change at all, but suffers from the limitations that 1) it doesn't support transformations of the interface to formats more suitable to particular client devices, and 2) it does not provide a layer of indirection underneath widget invocations.

The Mobisaic [35] and Dynamic Documents [16] projects support a HTML-based structure for varying, location-dependent interfaces. Our scheme generalizes these approaches by incorporating resource discovery and interoperability of different interface elements.

The Georgia Tech CyberGuide project [21] focuses on prototyping applications augmented with various positioning systems, potentially without communications at all. Using such an approach requires the devices to be manually adapted to new environments.

Our conception of a "proxy server" is based on the model expressed explicitly in the Berkeley Client/Proxy/Server model [10] and implicitly in other work [6, 2] that places application-level or network-level entities near, but not at, the endpoints of communications. This is another way of thinking about Active Networks [33], driven by end-to-end design principles [25]: push agents to as close to the endpoints as possible. This concept of leveraging the well-connected, computationally powerful side of the wireless link (via "proxies" or "agents") pervades mobility research. It is also driven by the growing availability of workstation farms [3] designed to provide compute resources for just such applications.

7 Future and Continuing Work

Our continuing work involves iterating over the design and investigating various implementation approaches.

We are in the process of augmenting another room, this time focusing on video cameras, monitors, and audio devices integrated with MBone session feeds. This will allow us to see how the architecture generalizes to new kinds of controllable objects.

The current implementation has been tested only in a local area environment; work is continuing as to the specifics of how such servers aggregate (with union and intersection operations) and their hierarchy.

We are working with building architects and engineers at the Center for the Built Environment[4] to incorporate devices such as the centralized heating and air conditioning, vents, fans, and temperature/humidity sensors into our system. This could allow users to close the environmental control loop and adapt areas in accord with user preferences as they move.

In general, mobiles may be allowed controlled access to CPU resources directly rather than configured services. This allows custom installation of "last-hop" network protocols, codecs, and security modules that are too compute-intensive to run on the end-client (e.g., for allowing the use of end-to-end session keys in an untrusted domain, for delta-encoding of data, or for deploying private handoff prediction.) This requires a management layer for implementing policy decisions granting access to bandwidth, disk, and CPU. It also requires a mechanism for securely delegating operations [19].

Queued RPC mechanisms [15, 5] support disconnection and link variability by incorporating application-managed messaging state. Queued asynchronous notification support is not incorporated into our system, but it should be.[5]

We wish to add additional functionality to our map application, including the ability to tie together physical geography to network connectivity. Servers could be are pinned to their location on the floorplan and the connectivity graph is automatically overlayed as it is discovered. Also to be added are the specifics of the administrative domains: overlaying the list of services available at groups of servers on the map, and extensions to illustrate hierarchy.

A full specification of the IDL grammar and UI generation for different platforms is work-in-progress.

[4]http://www.ced.berkeley.edu/

[5]On the other hand, applications should also be able to ignore failed RPCs rather than queuing them, a more appropriate paradigm for situations such as with A/V equipment interaction — the client interface is designed to express the current state of external processes and most RPCs can be mapped to idempotent operations.

8 Conclusions

We have presented an architecture for "universal interaction," allowing a device to adapt its functionality to exploit services it discovers as it moves into a new environment.

The key elements of the architecture we have developed include: 1) augmented mobility beacons providing location information and security features, 2) an interface definition language allowing exported object interfaces to be mapped to client device control interfaces, and 3) client interfaces that maintain a layer of indirection, allowing elements to be remapped as server locations change and object interactions to be composed into complex behaviors

We have also provided a detailed description of our prototype implementation of the architecture and a number of example services in actual use in the CS building at UC Berkeley.

Though we have focused on wireless, facets of our approach are applicable to wired networking. Examples include automatic reconfiguration for fault tolerance and scoping of access to services through limited broadcast of nonces.

9 Acknowledgements

Thanks to everyone in the BARWAN/Daedalus/GloMop projects for providing me with tools and maintaining our environment, especially Hari Balakrishnan, Venkat Padmanabhan, Elan Amir, and Mark Stemm. Richard Bukowski and Laura Downs provided the initial AMX control software. James Landay and Eric Brewer supplied ideas for extending this project. Mark Stemm and Steve McCanne provided detailed comments on a draft of this paper, and Steve also provided in-depth background material.

This work was supported by DARPA contract DAAB07-95-C-D154, the California State MICRO Program, Hughes, Metricom, Daimler-Benz, PCSI, and GTE.

References

[1] AMIR, E., MCCANNE, S., AND KATZ, R. Receiver-driven Bandwidth Allocation for Lightweight Sessions. *Proc. ACM Multimedia '97* (1997). To appear.

[2] AMIR, E., MCCANNE, S., AND ZHANG, H. An Application-level Video Gateway. *Proc. ACM Multimedia '95* (November 1995), 511–522.

[3] ANDERSON, T., PATTERSON, D., CULLER, D., AND THE NOW TEAM. A Case for Networks of Workstations: NOW. *IEEE Micro* (February 1995).

[4] AZIZ, A., AND DIFFIE, W. Privacy and Authentication for Wireless Local Area Networks. In *IEEE Personal Communications* (First Quarter 1994), IEEE, pp. 25–31.

[5] BAKRE, A., AND BADRINATH, B. Reworking the RPC Paradigm for Mobile Clients. *Mobile Networks and Applications 1*, 4 (January 1997), 371–86.

[6] BALAKRISHNAN, H., SESHAN, S., AMIR, E., AND KATZ, R. Improving TCP/IP Performance over Wireless Networks. In *Proc. 1st ACM Conf. on Mobile Computing and Networking* (San Franscisco, California, November 1995), pp. 2–11.

[7] BROWN, D. Techniques for Privacy and Authentication in Personal Communications Systems. In *IEEE Personal Communications* (August 1995), IEEE, pp. 6–10.

[8] BUKOWSKI, R., AND DOWNS, L. User's Guide to the 608-7 Computer Control System, December 1993. UC Berkeley CS260 class project.

[9] CLARK, D., AND TENNENHOUSE, D. Architectural Considerations for a New Generation of Protocols. *Proceedings of ACM SIGCOMM '90* (September 1990), 201–208.

[10] FOX, A., BREWER, E., GRIBBLE, S., AND AMIR, E. Adapting to Network and Client Variability via On-Demand Dynamic Transcoding. *ASPLOS* (1996).

[11] FOX, A., AND GRIBBLE, S. D. Security on the Move: Indirect Authentication using Kerberos. *Proceedings of the 2nd ACM International Conference on Mobile Computing and Networking* (1996), 10–12.

[12] GOLDBERG, I., WAGNER, D., THOMAS, R., AND BREWER, E. A Secure Environment for Untrusted Helper Applications: Confining the Wily Hacker. *Proc. of the 6th USENIX Security Symposium* (1996).

[13] HARTER, A., AND HOPPER, A. A Distributed Location System for the Active Office. *IEEE Network Magazine 8*, 1 (January 1994).

[14] IOANNIDIS, J., DUCHAMP, D., AND MAGUIRE, G. IP-Based Protocols for Mobile Internetworking. In *ACM SIGCOMM Symposium on Communications, Architecture, and Protocols* (1991), pp. 235–245.

[15] JOSEPH, A., DELESPINASSE, A., TAUBER, J., GIFFORD, D., AND KAASHOEK, M. F. Rover: A Toolkit for Mobile Information Access. *Proceedings of the Fifteenth Symposium on Operating System Principles* (December 1995).

[16] KAASHOEK, F., PICKNEY, T., AND TAUBER, J. Dynamic Documents. *Workshop on Mobile Computing Systems and Applications* (December 1994).

[17] KATZ, R. H. Wireless Overlay Networks. In *Proceedings 1996 SPIE Conference on Multimedia and Networking* (San Jose, California, January 1996).

[18] KATZ, R. H., BREWER, E. A., AMIR, E., BALAKRISHNAN, H., FOX, A., GRIBBLE, S., HODES, T., JIANG, D., NGUYEN, G. T., PADMANABHAN, V., AND STEMM, M. The Bay Area Research Wireless Access Network (BARWAN). In *Proceedings Spring COMPCON Conference* (1996).

[19] KOTTMAN, D. A., WITTMANN, R., AND POSUR, M. Delegating Remote Operation Execution in a Mobile Computing Environment. *Mobile Networks and Applications 1*, 4 (January 1997), 387–98.

[20] LIU, G., AND MAGUIRE JR., G. A Class of Mobile Prediction Algorithms for Wireless Mobile Computign and Communications. *Mobile Networks and Applications 1*, 2 (October 1996), 113–122.

[21] LONG, S., KOOPER, R., ABOWD, G., AND ATKENSON, C. Rapid Prototyping of Mobile Context-Aware Applications: The CyberGuide Case Study. *Proc. 2nd ACM Conf. on Mobile Computing and Networking* (November 1996), 97–107.

[22] MACINTYRE, B., AND FEINER, S. Future Multimedia User Interfaces. *Multimedia Systems Journal 4*, 5 (October 1996), 250–268.

[23] MCCANNE, S., JACOBSON, V., AND VETTERLI, M. Receiver-driven Layered Multicast. *ACM SIGCOMM '96* (August 1996), 117–130.

[24] PERKINS, C. *IP Mobility Support*. RFC, Oct 1996. RFC-2002.

[25] SALTZER, J., REED, D., AND CLARK, D. End-to-end Arguments in System Design. *ACM Transactions on Computer Systems 2*, 4 (1984), 195–206.

[26] SCHILIT, B. N., ADAMS, N., GOLD, R., TSO, M., AND WANT, R. The PARCTAB mobile computing system. In *Proceedings Fourth Workshop on Workstation Operating Systems (WWOS-IV)* (October 1993), IEEE, pp. 34–39.

[27] SCHILIT, B. N., ADAMS, N. I., AND WANT, R. Context-Aware Computing Applications. In *Proceedings of the Workshop on Mobile Computing Systems and Applications* (December 1994), IEEE Computer Society, pp. 85–90.

[28] SCHILIT, B. N., AND THEIMER, M. M. Disseminating Active Map Information to Mobile Hosts. In *IEEE Network* (Sept/Oct 1994), IEEE, pp. 22–32.

[29] SCHILIT, W. *System Architecture for Context-Aware Mobile Computing*. PhD thesis, Columbia University, 1995.

[30] SMITH, B., EOLAS TECHNOLOGIES, INC., COMPUTERIZED PROCESSES UNLIMITED, INC., AND ROSEMAN, M. DpTcl.tcl, 1996. Obtained as alpha code.

[31] STEMM, M., AND KATZ, R. H. Vertical Handoffs in Wireless Overlay Networks. *ACM Mobile Networking (MONET), Special Issue on Mobile Networking in the Internet* (Fall 1997). To appear.

[32] SUKAVIRIYA, P., FOLEY, J., AND GRIFFITH, T. A Second Generation User Interface Design Environment: The Model and The Runtime Architecture. *Proceedings of Human Factors in Computing Systems '93* (April 1993), 375–382.

[33] TENNENHOUSE, D., SMITH, J., SINCOSKIE, W., WETHERALL, D., AND MINDEN, G. A Survey of Active Network Research. *IEEE Communications Magazine* (January 1997), 80–86.

[34] VEIZADES, J., GUTTMAN, E., PERKINS, C., AND KAPLAN, S. *Service Location Protocol Internet Draft #17, draft-ietf-svrloc-protocol-17.txt*. IETF, 1997.

[35] VOELKER, G., AND BERSHAD, B. Mobisaic: An Information System for a Mobile Wireless Computing Environment. *Workshop on Mobile Computing Systems and Applications* (December 1994).

[36] WATSON, T. Application Design for Wireless Computing. In *Workshop on Mobile Computing Systems and Applications* (Dec. 1994), IEEE Computer Society.

[37] WEISER, M. Some Computer Science Issues in Ubiquitous Computing. *Communication of the ACM 36*, 7 (July 1993).

[38] WOOD, K. R., RICHARDSON, T., BENNETT, F., HARTER, A., AND HOPPER, A. Global Teleporting with Java: Toward Ubiquitous Personalized Computing. *IEEE Computer 30*, 2 (1997), 53–59.