Pervasive Computing: What Is It Good For?

Andrew C. Huang
ach@cs.stanford.edu

Benjamin C. Ling
bling@cs.stanford.edu

Shankar Ponnekanti
pshankar@cs.stanford.edu

Armando Fox
fox@cs.stanford.edu

Abstract

The first mass-produced pervasive computing devices are starting to appear—the AutoPC, the Internet-connected ScreenFridge, and the combination Microwave Oven/Home Banking terminal. Although taken separately they appear bizarre, we believe they will play an important role in a world of pervasive computing. Specifically, these devices will accept or deliver information in the context in which it will be most useful, decoupling the information from the context in which it was originally created. We describe an extensible and modular architecture called *Rome* (to which all roads lead) that addresses this information-routing problem while leveraging significant existing work on composable Internet services and adaptation for heterogeneous devices. Rome's central abstraction is the concept of a *trigger*, a self-describing chunk of information bundled with the spatial and/or temporal constraints that define the context in which the information should be delivered. The Rome architecture manages triggers at a centralized infrastructure server and arranges for the triggers to be distributed to pervasive computing devices that can detect when the trigger conditions have been satisfied and alert the user accordingly. The main contribution of the architecture is an infrastructure-centric approach to the trigger management problem. We argue that pervasive computing devices benefit from extensive support in the form of *infrastructure computing services* in at least two ways. First, infrastructure adaptation services can help manage communication among heterogeneous devices. Second, access to public infrastructure services such as MapQuest and Yahoo can augment the functionality of trigger management because they naturally support the time- and location-dependent tasks typical of pervasive-computing users. We describe our experience with a functional prototype implementation that exploits GPS to simulate an AutoPC, and propose a research agenda to further the creation and deployment of pervasive-computing infrastructure based on the architecture we describe.

To appear in the International Workshop on Data Engineering for Wireless and Mobile Access (MobiDE) in conjuction with ACM MobiCom '99, Seattle, WA, April 1999

1 The Problem Is That There Is No Problem

The Internet-connected Electrolux ScreenFridge [8], the NCR Microwave Bank [2], and the new AutoPC [1] appear to be primitive first steps in the direction of pervasive computing. If these efforts sound a bit outlandish, there's a good reason: the devices are solutions in search of a problem.

Yet the devices do have something in common with Web browsers, pagers, cell phones, grocery lists, and to-do notes stuck on the door. We are inundated with information of all kinds, arriving over various media, and targeted for various tasks—do this tomorrow, check up on that when you're in the office, call me back, and so on. This suggests that one natural target for pervasive computing is *data management*—getting information into the temporal or spatial *context* in which it will be most useful (as opposed to the context in which the information was originally created), and using pervasive computing devices to accept or deliver it.

It has been argued [11, 14] that pervasive computing will have succeeded when computers "disappear into the infrastructure" and we find ourselves using computer-assisted task-specific devices, as opposed to computing devices *per se*. In this paper we propose a research program whose goal is the useful incorporation of such devices into the environment as data managers. In particular, we describe a simple pervasive computing architecture to address the above problem, along with an implemented prototype using off-the-shelf PC's and pervasive computing devices.

We take an infrastructure-centric point of view: like PDA's and handheld devices, pervasive computing devices will be fundamentally dependent on infrastructure "glue" in order to be truly useful. For PDA's, the original rationale behind this assertion was the need for adaptation: since a primary application of the devices is information retrieval from the Internet, infrastructure support is needed to adapt these devices to a network infrastructure not designed for them [5]. The analogous argument for pervasive computing is that humans receive and deal with information in a variety of temporal and spatial contexts, and although pervasive computing devices are useful as "end-unit" sensors and actuators to assist with information management tasks, infrastructure support is needed to tie them together and address the distributed information management problem.

1.1 A Motivating Example

Consider the following scenario:

Opening your refrigerator to take out a soda, you notice that there is only one can left. You scan its UPC with the scanner attached to your refrigerator. This action adds soda to your shopping list. You plan to have guests over this weekend, and make a note on your ScreenFridge that you need to replenish your supply of drinks by Friday.

The next day, on your drive home from work, you happen to approach a local supermarket. Your GPS-enabled AutoPC, previously informed by your refrigerator that purchases need to be made, signals that you are near a grocery store, and if it is convenient, that you should stop by the supermarket on the way home. Suppose you do not act on the opportunity, and Friday rolls around and you still have not visited the supermarket; in this case, a message to buy drinks is sent to your pager, or an alarm is triggered in your PDA, or both.

The key observation that follows from the above example is that information is rarely useful at the time and place it is generated (in this example, you generate the information when you remove the last soda from the refrigerator). Rather, the information must be re-presented later, where and when it can be acted on. That is, information is most useful when it is delivered in the correct temporal or spatial context (when you are near a supermarket, or when a visit from guests is imminent).

1.2 No Computer Is An Island

This simple example illustrates two important ways in which pervasive computing relies on infrastructure support. The first, and obvious, dependence is for communication: items are added to the "shopping list" by your refrigerator but the list itself needs to be accessible elsewhere (for example, on your PDA as you cruise the aisles in the supermarket, or ideally, in a world of truly pervasive computing, on supermarket terminals or in your augmented-reality goggles [9]). The information has to be transported between devices.

The less obvious but more important dependence is on *infrastructure services*. By these we refer to (usually) publicly-accessible interactive services that perform on-demand computation over large datasets. In the context of the present work, an infrastructure service is simply one that does not run on any of the pervasive computing devices themselves—it may run in the public Internet or in a private home-area network. Infrastructure Services may be necessary for any of several reasons:

- **Unwieldy datasets:** One example of an unwieldy dataset is a UPC database. In the example given in the previous section, the refrigerator may need to translate the UPC to a product name so that the shopping list is readable by humans. Although the refrigerator might cache the UPC's of frequenly-purchased products, a general UPC translation service would necessarily be infrastructural.
- **Rapidly changing data:** Timely information, such as traffic reports, news, and stock quotes, changes too rapidly to make continuous distribution practical for intermittently-connected users. Furthermore, devices such as PDA's and the AutoPC may not have constant connectivity; therefore, such services require infrastructure support.
- **Computation:** The computation required to generate a result may exceed the storage or CPU resources available to a small device. For example, some online mapping services such as MapQuest support queries of the form "Given a starting location, find the geographically closest businesses of a given type."

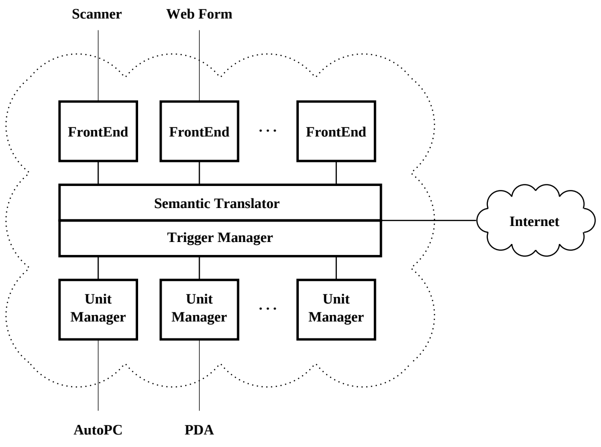

Figure 1: Overall Rome Architecture

Popular examples of Web-based infrastructure services that have the above properties include mapping and driving directions services, zip code lookup, search engines and business directories, up-to-the-minute financial information, and online auctions. Several aspects of our motivating example assume the ability to leverage such services: translating UPC's to product names, locating a supermarket through a directory service such as Yahoo!, converting the address to GPS coordinates ("geocoding") through programmatic interfaces to mapping services such as MapQuest, and possibly sending a text-based, on-demand notification to a pager or similar device through an Internet gateway, such as SkyTel.

Recognizing the need for infrastructural support for pervasive computing devices, we present Rome, a uniform architecture for addressing the information management and infrastructure dependence issues exposed by the example given in the previous section. In the following sections, we describe the Rome architecture, discuss our prototype implementation in progress, and show how this approach promises rapid deployment by leveraging other existing research, both in design philosophy and implementation.

2 The Architecture

2.1 Triggers

The notion of a trigger is central to Rome. We use the term *trigger* to mean information paired with its useful context. Conceptually, a trigger is some action that should be taken when a previously stated condition is satisfied. The trigger is not novel to Rome; it has been in use in the database community for years. However, Rome extends the classical database notion of a trigger by allowing decentralized evaluation of triggers.

Triggers are pushed to end devices, such as PDA's or AutoPC's, allowing evaluation to occur at those devices. Although in some cases evaluation could occur at a central location such as a server, in other cases the end device is the only appropriate device to evaluate the trigger, because information such as location may be local to the device and not known by others. Rome provides the ability to (a) determine which devices should receive which triggers, (b) route the selected triggers to the devices, and (c) possibly assist in determining when to fire them (in case some trigger terms cannot be evaluated locally on the device).

The Rome architecture is shown in Figure 1. The system is well-connected to the Internet infrastructure (we illustrate shortly why this is useful) and consists of several components. *FrontEnds* are used for input into a *Semantic Translator*, which in turn interfaces with a *Trigger Manager*. Also, a *Unit Manager* exists for each type of end-device. The function of each component is described in Section 3.1.

2.2 Trigger Management

Conceptually, a trigger is composed of a condition and an action. When the condition evaluates to true, some action should occur.

Typically, the desired goal is the completion of some high-level task (e.g. buy drinks). The high-level task is translated by the system's Semantic Translator to a corresponding set of triggers. Some tasks may be represented as a single trigger, such as an alarm for a specific time. Others may be represented as a combination of several triggers. In the supermarket example, the high-level task can be mapped to two distinct triggers: 1) A trigger that is fired when a certain spatial constraint is satisfied (e.g. I am near a supermarket), or 2) a trigger that is fired when a certain temporal context is met (e.g. it is now Friday 3 p.m. and I have to buy drinks before the party). A *term* is a primitive that evaluates to true when a particular spatial or temporal constraint is satisfied.

All triggers are stored in the Rome system; a subset is forwarded to those devices that can evaluate the trigger's condition. For example, location-sensitive triggers may be transferred to a GPS-enabled AutoPC since such a device can best evaluate the user's location. In contrast, the best place for purely time-based triggers might be a PDA that the user always carries. When the high-level task has been acted on, notification is sent to Rome so that all other triggers associated with that piece of information can be removed, in order to avoid redundant messages on tasks that have already been acted on.

2.3 An Illustrated Example

Returning to the supermarket example, we describe the chain of events in relation to the proposed architecture:

1. A user scans the UPC of a bottle, and enters a requirement to buy drinks by Friday. The scanner sends this information to the Rome FrontEnd designed for the scanner's interface. In general, for every input format, there is a FrontEnd that collects information about high-level tasks.

2. The FrontEnd then forwards the information to the Semantic Translator which translates high-level tasks into sets of triggers. Since Rome is well-connected to the Internet, it can retrieve pertinent information such as the location and hours of local supermarkets using a publicly-accessible service such as Yahoo or MapQuest. Using this information, the translator produces two triggers from the high-level task "Buy drinks by Friday":

   - Trigger 1:
     - Condition: (location $\in R) \wedge (t > T_1) \wedge (t < T_2)$
     - Data: "Since you are driving home from work and passing by a grocery store at location R, you could stop to buy drinks for Friday."
   - Trigger 2:
     - Condition: $(t = T)$
     - Data: "Buy drinks today."

   Note that the condition in Trigger 1 is composed of three terms: (location $\in R$), $(t > T_1)$, $(t < T_2)$. The time terms represent the period of time associated with the user's drive home from work. In general, a condition can be composed of an arbitrary number of terms, connected by logical operators.

3. Rome then inspects its list of accessible end-devices; each of the end-devices and its corresponding properties were made known to Rome, by previous registration. Hence Rome is aware of information such as processing power and connectivity of each end-device, and is capable of distributing triggers to end-devices. The first trigger above is be routed to an AutoPC equipped with GPS, and the second trigger routed to a PDA.

4. Each end-device stores the trigger(s) assigned to it, and when the appropriate context is satisfied, renders the data to the user. For example, when the AutoPC nears a supermarket during open hours, the user is informed of the event, perhaps by the AutoPC's text-to-speech facility. Because Rome knows each device's capabilities, it can ensure that each device only receives data that it can suitably render.

5. When the event is acted on, the user will indicate that the high-level task has been accomplished to the AutoPC, which will in turn inform Rome. Because the AutoPC is weakly connected, it can notify Rome via email that the task has been acted on. Rome can then proceed to cancel other triggers that were created from the same high-level task, as they are now unnecessary. The next time other devices communicate with Rome, extraneous triggers will be deleted from those devices as well.

3 Prototype Implementation

3.1 General Overview

Triggers are the mechanism by which reminders are produced in a context where the information is useful. A trigger object contains trigger data, condition, and an optional special handler, which replaces the default handler if special actions are to be taken. (In the current prototype, only default handlers are supported.) The user directly enters trigger information into the Rome System, which determines which devices are capable of processing specific triggers and distributes the triggers accordingly.

We use the term *Mobile Information Appliance* (MIA) to refer generically to a device that can act as the endpoint of a trigger.

Rome comprises the following components:

The Frontend provides an interface through which the user can enter trigger information—conditions to be met for the trigger to go off and the actions that need to be taken when the trigger goes off. Although the prototype supports only Web form entry of trigger conditions directly (eliminating the need for a Semantic Translator), keeping the frontend separate from the Trigger Manager allows incremental addition of more sophisticated input mechanisms, e.g. a simulated ScreenFridge.

The Trigger Manager runs in the infrastructure and stores all the triggers entered by the user. This component must be up and accessible all the time. This forwards the triggers to the appropriate endpoints upon request using the various unit managers.

The Unit Manager is an MIA specific component residing in the infrastructure. It hides the peculiarities of each type of MIA from the Trigger Manager and helps in transferring triggers from the Trigger Manager to the MIA's.

The Trigger Acceptor resides on the MIA and communicates with the appropriate unit manager in the infrastructure to download triggers if and when necessary.

The Trigger Handler resides on the MIA and is responsible for evaluating the trigger condition and executing the appropriate handler. The Trigger Handler consists of three elements. *T*erm Evaluators interface with the device clock or GPS receiver to evaluate the temporal and spatial terms in the condition; when a term changes in value, a callback is sent to the Condition Evaluator. The *C*ondition Evaluator re-evaluates the Boolean expression of terms each time a callback from the Term Evaluator is received; when the expression evaluates to true, the Data Handler is invoked to renders the trigger's data to the user using whatever notification mechanism is appropriate to the MIA (window popup, text-to-speech conversion, alarm sound, etc.)

3.2 Implementation Details

We have leveraged several existing "off the shelf" software tools and technologies for putting together the system: Ninja [6], TeleType software [12] (for GPS), and Microsoft Windows CE. We are using a GPS enabled Philips Nino (running WinCE), which we expect to replace with an AutoPC in actual use.

In the current implementation, the MIA (Nino) communicates with the Unit Manager using the standard Windows CE serial cradle. The Trigger Acceptor, a piece of native WinCE code that runs on the Nino, initiates a connection to the Unit Manager, a Ninja ISPACE service running in the infrastructure. Triggers are transferred between the Trigger Acceptor and the Unit Manager using a simple TCP-based serialization protocol. In a real implementation, we expect to use a two-way wireless connection rather than the cradle and allow for the possibility of proactively pushing triggers to an MIA capable of asynchronous receiving (rather than waiting for an explicit request from the MIA). The Unit Manager gets the triggers from the Trigger Manager, another Ninja ISPACE service that maintains a database of trigger objects in an XML-like serialized form.

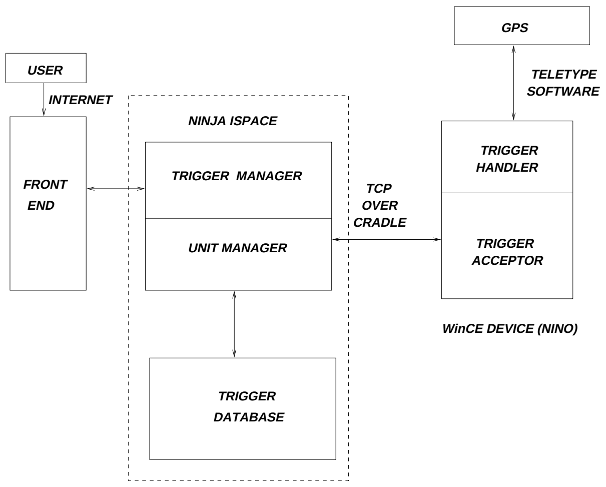

Figure 2: Rome System

The Trigger Handler on the WinCE device converts from the internal representation of the trigger to a C++ trigger object. The term evaluators in the Trigger Handler access the GPS data and register for time-based notifications via the WinCE API.

Although primitive, the prototype demonstrates the feasibility of the Rome system. We implemented the motivating example scenario using this prototype: the user enters a trigger ("buy drinks") through the frontend asking the system to remind him if he is near the supermarket in a specified time interval, and when the GPS coordinates indicate that the user is close to the supermarket and the current time is within the specified limits, a window pops up on the Nino. The high degree of modularity of the prototype should allow us to improve it by incrementally adding more sophisticated user input mechanisms (e.g. ScreenFridge simulator) and incorporating new devices (e.g. AutoPC).

4 Related Work

4.1 Infrastructure Computing

The UC Berkeley TACC [3] framework enables the creation of interactive Internet applications by composing modules that either implement new functionality or leverage the functionality of another already-deployed remote service. For example, an early TACC application was a “meta-search” engine that queried various existing search engines (using HTML screen scrapers) and ranked the collated results; this application could be composed with a thin-client browser adaptor [4] to deliver an efficient multi-engine search service to a handheld PDA device with a limited user interface. In this sense, TACC was a primitive attempt to provide a framework for *programming* the Internet using existing services as building blocks. The Java-based Ninja [6] framework improves on TACC by providing strongly-typed composable programmatic interfaces for service modules (thus enabling automatic composition) and providing a core set of services and primitives for building and composing modules (service registry, module replication with transparent failure-tolerant remote invocation, authenticated RMI, etc.) We used Ninja as a starting point for building Rome, since the Rome architecture is consistent with the trend toward infrastructure support for wide-area services and diverse devices.

The need for interoperability among diverse devices and heterogeneous networks has been extensively addressed via work on transformational and other adaptation proxies. In the context of the present proposal, for example, an AutoPC can render audio (text-to-speech), whereas a pager or PDA has limited text and graphics rendering. TACC demonstrated the value of putting transformation services into the infrastructure; since the Rome Trigger Manager already acts as a centralized resource in our pervasive computing architecture, it is a natural place to collocate transformation capability.

4.2 Pervasive Computing

In *The Design of Everyday Things* [10], Donald Norman advocates moving information “into the world” as a sound design technique for everyday objects. Pervasive computing using the Rome architecture is a concrete instantiation of this principle: at the moment a useful piece of information is created, it can be routed (perhaps via store-and-forward) to an information management system whose responsibility is to ultimately deliver it into the physical context in which it will be useful.

In Weiser and Want’s seminal Tab system [14], the wearable devices called Tabs (later known as Active Badges) relied on extensive infrastructure support to provide functionality such as tracking a user around a building, automatically opening doors for her, forwarding her calls to the phone nearest her current location, and having a terminal retrieve her preferences when she sits down to use it. We believe the intimate connection between truly pervasive computing and strong infrastructure support, as demonstrated in the Tab system, is fundamental—hence our strategy of combining a flexible and robust framework for “programming the infrastructure”, i.e. Ninja, with a novel data-management architecture whose central abstraction is the context-sensitive trigger and whose primary responsibility is the creation, management, and routing of triggers, with infrastructure support.

Steve Mann’s head-mounted augmented reality system [9] allows a computer-generated image (edge detected and partly texture mapped) to be superposed with the “real” image that is seen through the glasses. Among other things, the overlaid image can be used to annotate the virtual counterparts of real-life objects; for example, a virtual “buy soda” sign hung on the supermarket can be seen as the user passes by the supermarket while wearing the glasses. By treating the glasses as a device capable of instantiating a location-based trigger, Rome provides a unified framework for supporting such reality augmentation devices.

Recent work on smart homes and smart spaces [7] has focused on controlling the components of a room using an extensible architecture in which user interfaces to objects in the room are dynamically generated and presented to the user, based on static documents describing object attributes. Again, the commonality with Rome is the reliance on infrastructure “glue” to bring the devices together. Rome can enhance the “smartness” of smart spaces by coupling automatic actuation with the correct context; for example, your alarm clock might go off half an hour earlier than usual, based on its knowledge of a meeting you have scheduled (it has access to your personal calendar) and the report of an accident on your route to work (it has access to Web-based traffic information).

4.3 Triggers

Triggers have been in use in the relational database community since at least SQL version 3 [13]. In that context, a trigger is a specification of some actions to take whenever a modification to a database table results in some constraint on the relation being met. We have extended the basic trigger in two ways. First, recognizing that dynamic attributes such as location and time are fundamental to our task of moving information to the right context, we have provided the ability to instantiate triggers based on these attributes. We rely on the specific abilities of pervasive-computing devices to accomplish this. Second, we allow partial and distributed evaluation of triggers. Specifically, we look for affinities between terms in the trigger condition and the specific abilities of each device (GPS, time based alarm, etc.)

5 Research Agenda

With the basic framework of the initial prototype complete, we will begin to address some of the following issues:

5.1 Handling multiple users

Most information transmitted to and from pervasive devices is likely to be private by default. However, we believe many users will prefer to avoid the complexity of operating a Home Base in their home, delegating this instead to an Internet data center or comparable infrastructure installation. Note that many people already use public infrastructure for email and gatewaying messages to pagers, so the idea of using public infrastructure for private data is not without precedent. We intend to build Rome as a shared infrastructure for private data, but leave open the possibility to extend the system to support group-accessible triggers. This idea is similar to group-accessibility in file systems; in a sense, triggers, like files, are write-shared across multiple users and consistency must be maintained across users.

5.2 Trigger consistency

Sharing triggers across multiple users and devices introduces the problem of trigger consistency. In many cases, when triggers are satisfied or cancelled, the trigger should be removed from the system; this includes removing the trigger from any devices on which it resides. Maintaining consistency is further complicated by the fact that not all devices are constantly connected to the infrastructure.

5.3 User interface

In the initial prototype, triggers are entered using a Web-form interface. If Rome is to be widely accepted, other interfaces must be designed to make it easy for users enter trigger information. As the motivating example suggests, one possible interface is the UPC scanner on the ScreenFridge; this interace is an easy way to enter "shopping-list" reminders, but it is clearly limited to the particular context. Therefore, architecture is designed so that any number of such interface modules, or Frontends, can easily be added.

5.4 Semantic translation

When easier-to-use interfaces are integrated into the system, there must be a mechanism to translate high-level "requests" such as scanning an item on a UPC scanner into a set of primitive triggers. In many cases, a user-preferences file will aid the translation; such a file could include locations of interest (e.g., grocery stores within a certain radius of a user's home), the user's default schedule, and a link to the user's actual schedule. In keeping with the composable applications philosophy, we are making this translation mechanism separable from and composable with the Trigger Manager to allow maximum flexibility for future work in this area.

5.5 Multi-term conditions

Since certain condition parameters, such as those that depend on rapidly-changing information, are best evaluated in the infrastructure, there may exist conditions where some terms should be on the device and some terms should be evaluated at the Rome trigger manager. As was the case with trigger consistency, this problem is complicated by the fact that some devices are only intermittently connected.

6 Conclusions

The architecture described in the previous sections brings coherence to a number of other problems whose relationships to each other were previously not obvious. It provides a framework that naturally accommodates (and increases the utility of) such diverse devices as the AutoPC, the ScreenFridge, and augmented reality glasses. The design of this architecture was heavily influenced by the following observations:

- Many of these task-specific devices are best suited for data management tasks. The Rome architecture attempts to capture a large class of such tasks by extending the database trigger concept with first-class abstractions for spatial and temporal conditions and by allowing partial and distributed evaluation.

- Infrastructure support increases the utility of pervasive computing devices in at least two ways: by providing connectivity among heterogeneous devices, and by leveraging existing infrastructure services such as geocoding and queries against very large datasets. For this reason, Ninja is used to provide a set of services that have composable programmatic interfaces.
- Device heterogeneity does not appear to be decreasing any time soon. In fact, the ubiquitous-computing goal of a broad array of task-specific devices rather than general-purpose computers leads us to expect heterogeneity to increase further. Therefore, Rome strongly leverages prior work on adapting to device and network heterogeneity.

As we move toward the widespread integration of diverse computing resources—PC's, handhelds, mobile wireless devices, pervasive computing devices—an architectural approach is required that leverages existing work, is extensible enough to accommodate future developments, and describes a meaningful paradigm for interaction among such devices. Rome embodies these qualities, is implementable today using currently available infrastructure software tools and pervasive computing devices, and is modular and extensible enough to track future developments in pervasive computing technology.

7 Acknowledgments

We thank Dr. Kathy Richardson of Compaq NSL and the MobiDE anonymous referees for their helpful feedback on Rome and on this paper. Hongbin Qi, Li-Cheng Wang, and Mark Wang contributed to the prototype implementation.

References

[1] Clarion Corp. AutoPC announcement, 1999. `http://www.autopc.com/walkthrough/communication/`.

[2] NCR Corp. NCR microwave bank announcement, 1999. `http://www.wired.com/news/news/technology/story/14949.html`.

[3] Armando Fox. The Case for TACC: Scalable Servers for Transformation, Aggregation, Caching and Customization. Qualifying Exam Proposal, UC Berkeley Computer Science Division. `http://www.cs.berkeley.edu/~fox/papers/quals.ps`, April 1997.

[4] Armando Fox, Ian Goldberg, Steven D. Gribble, Anthony Polito, and David C. Lee. Experience with Top Gun Wingman: A proxy-based graphical web browser for the Palm Pilot PDA. In *IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware '98)*, Lake District, UK, September 15–18 1998.

[5] Armando Fox, Steven D. Gribble, Yatin Chawathe, and Eric A. Brewer. Adapting to network and client variation using active proxies: Lessons and perspectives. *IEEE Personal Communications (invited submission)*, Aug 1998. Special issue on adapting to network and client variability.

[6] Steven D. Gribble, Matt Welsh, Eric A. Brewer, and David E. Culler. The multispace: An evolutionary approach to internet-scale services. In *Second USENIX Symposium on Internet Technologies and Systems (USITS 99)*, Aug 1999.

[7] Todd D. Hodes and Randy H. Katz. Enabling "smart spaces:" entity description and user interface generation for a heterogeneous component-based distributed system. In *DARPA/NIST Smart Spaces Workshop*, Gaithersburg, MD, July 1998.

[8] Electrolux Inc. Electrolux screen fridge (news article), 1999. `http://www.wired.com/news/news/email/explode-infobeat/technology/story/%17894.html`.

[9] Steve Mann. Wearable computing: A first step toward personal imaging. *IEEE Computer*, 30(2), Feb 1997.

[10] Donald A. Norman. *The Design of Everyday Things*. Doubleday, 1990.

[11] Donald A. Norman. *The Invisible Computer: Why Good Products Can Fail, The Personal Computer Is So Complex, and Information Appliances Are the Solution*. MIT Press, 1998.

[12] TeleType Corp. TeleType GPS receiver and software. `http://www.teletype.com`.

[13] Jeffrey D. Ullman and Jennifer Widom. *A First Course in Database Systems*. Prentice-Hall, 1997.

[14] Mark Weiser. The computer for the twenty-first century. *Scientific American*, pages 94–100, September 1991.