UNIVERSITY OF CALIFORNIA, SAN DIEGO

UbiBot: A System for Experimenting with Mobile Devices on a Wireless Network

A thesis submitted in partial satisfaction of the
requirements for the degree Master of Science

in

Computer Science

by

Erwin Abad Vedar

Committee in charge:

Professor William Griswold, Chair
Professor James Hollan
Professor Ranjit Jhala

2011

The Thesis of Erwin Abad Vedar is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

Chair

University of California, San Diego

2011

DEDICATION

I'd like to dedicate this thesis to my advisor, Bill Griswold. He's always encouraged me to continue and finish what I was doing, even in my moments of doubt. This work would not have been possible without his academic leadership and patience with his busy student.

TABLE OF CONTENTS

List of Abbreviations

MCAC	mobile context-aware computing
LBIM	Location-Based Instant Messaging
LBR	Location-Based Reminders
LBAT	Location-Based Automated Tour Guide

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to acknowledge the guidance and support of Professor Bill Griswold. His encouragement and patience have been invaluable.

I would also like to acknowledge W. Brian Evans for continuing the work of UbiBot beyond the intital proof-of-concept stage. His efforts enabled students not affiliated with the project to expand upon it. Also, I'd like to thank John Egan and Mark Gahagan for their efforts in developing the Enhanced Location-Based Reminders service.

Chapters 1, in part, is a reprint of the material as it appears in "UbiBot – Prototyping Infrastructure for Mobile Context-Aware Computing" in *Proceedings of the Second Workshop on Pervasive Computing Education (PerEd '09)*. Erwin Vedar, W. Brian Evans, William G. Griswold, 2009. The thesis author was the primary investigator and author of this paper.

Chapters 2, in part, is a reprint of the material as it appears in "UbiBot – Prototyping Infrastructure for Mobile Context-Aware Computing" in *Proceedings of the Second Workshop on Pervasive Computing Education (PerEd '09)*. Erwin Vedar, W. Brian Evans, William G. Griswold, 2009. The thesis author was the primary investigator and author of this paper.

Chapters 3 is, in part, is a reprint of the material as it appears in "UbiBot – Prototyping Infrastructure for Mobile Context-Aware Computing" in *Proceedings of the Second Workshop on Pervasive Computing Education (PerEd '09)*. Erwin Vedar,

W. Brian Evans, William G. Griswold, 2009. The thesis author was the primary investigator and author of this paper.

ABSTRACT OF THE THESIS

UbiBot: A System for Experimenting with Mobile Devices on a Wireless Network

by

Erwin Abad Vedar

Master of Science in Computer Science

University of California, San Diego 2011

Professor William Griswold, Chair

Web 2.0 technologies have fueled a new generation of applications that come to the desktop from the network. The emerging field of mobile context-aware computing (MCAC) would benefit from network-based applications even more than desktop computing. With MCAC, there are many issues that a network application infrastructure needs to address beyond providing mere functionality, such as low network speed and robustness, small, battery-powered devices, and limitations in the

software these devices are able to run. Furthermore, these devices offer unique sensing capabilities such as microphones, cameras, and GPS. Taking full advantage of these in network-based applications requires more flexibility than simply providing computing and network utilities.

We introduce UbiBot, an extensible system for experimenting with network-based services for the mobile. UbiBot addresses many of the problems of mobile computing by employing a publish-subscribe architecture that enables dynamically reconfiguring the system to incorporate new services, delegate computation, and manage network performance issues, yet without having to modify the software on the mobile devices. Furthermore, the software for the mobile can be adapted to the evolving capabilities of new devices. We demonstrate the flexibility and ease of UbiBot through several case studies.

CHAPTER 1: INTRODUCTION

1.1 Motivation

The emerging field of mobile context-aware computing (MCAC) can benefit from experimentation to accelerate development in the area, as well as support research and education. However, there are many challenges to experimentation with mobile context aware software. Mobile devices are small and battery-powered, and the software that they are able to run is limited in comparison to their desktop counterparts. Experimenters need to account for these resource constraints, or risk creating software that is unusable, or undesirable because it drains the device of battery or computation resources. Furthermore, the network connections for these devices, compared to a desktop machine wired into the network, are relatively low speed and not very robust. Experimenters cannot assume a constant connection to the network, or a static arrangement of clients and servers. These issues are further frustrated in research and education settings, where timelines are tight and such issues can derail the investigation into the topic of interest. Unless the investigation at hand is into network issues or mobile device limitations, these barriers hinder the progress of that investigation. Ideally, students could prototype a basic application that validates the basic functionality, and then add support for the above mobility considerations later in development, once the basic features had been worked out.

1.2 Mobile Wireless

The problem space of mobile computing is characterized by distributed systems in which participants connect and disconnect during operation, possibly at a different access point [Gad08], [Far04], [Cap03]. Thus the challenge in solving problems for this space is to create systems and applications that can function in a dynamic environment, subject to these changing connections within the system [Dav04]. The mobile computing space includes interactions over wireless networks. Connections to wireless networks are naturally mobile, with weak connections [Far04]. Creating applications for mobile devices such as smart phones falls into this problem space. As users move around with their devices, they may move out of range of one network access point while moving into range and reconnecting through another access point.

Smart phones are a common type of mobile device. Smart phone research and development has drawn much attention because of the devices' evolving capabilities and the concomitant increase in business potential [Dav04]. The number and popularity of such devices continues to increase [Dav04], [Sam01].

However, many barriers have to be overcome in order to reap the benefits offered by smart mobile phones, or more generally, mobile devices. The devices are characterized by small screens, low memory capacity and computational power [Muh04], [Dav04], [Gad08]. Their connections to the network are spotty and have

limited bandwidth [Far04], [Dav04], [Muh04], [Anc02]. While they pose much potential in terms of profitability and novel modes of interaction, these hurdles must be overcome to reach that potential. As the computation and interaction capabilities of mobile devices continue to grow, their limitations still need to be handled in a way that is acceptable to the user and easy for the developer.

The problems created by mobility can be reformulated as the need to create a meaningful, stable façade for the user that handles the limitations of the device and the network in the background. That is, users expect that their devices simply work as they move around; the connection and disconnection to the network should have as little consequence for the user as possible. These issues distract them from the task at hand. If the application is unable to function once partitioned from the network, it should still handle the situation in a graceful way. Also, cooperating applications need to be able to find each other in the network, regardless of their host device or the device's physical location in the network.

If the problems of mobility are sufficiently hidden from the user, then adoption of mobile devices will be a natural transition from the desktop.

1.3 Context Awareness

Mobility, as defined above, implies the possibility that the environment that the device is used in may change during operation. This yields different computing "contexts". Dey defines context as "Any information that can be used to characterize

the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves" [Dey01]. Context-aware applications don't purely rely on the user for input, but take information gathered from sensors in order to make decisions, possibly in an automated way [Du08].

Context awareness can potentially "enable a new breed of applications and services which would extend the functionality of human's subconciousness, being able to provide or act at the right time, with the right level of information" [Dev07]. In effect, this would "make computing devices smarter and more thoughtful for users" [Du08]. Context awareness, when used correctly, can make applications both more useful and easier to use. For example, a map application is more useful if the user's location on the map can be automatically calculated using GPS and displayed at the right place on the map. The user doesn't have to do any extra data entry, and the application can display information and interact with the user in a way that's relevant to the user's current situation. For example, if the user is looking for the nearest gas station, the application can limit its search results to the region closest to the user.

Context awareness presents many challenges to be overcome before these smart applications can be realized. Many have to do with converting raw sensor information to a form that is usable by the application and user. For example, GPS signals have to converted to GPS coordinates, but GPS coordinates aren't necessarily useful to the common user, who would better understand a street address. Furthermore, for a given context, there can be multiple, distributed, unconventional,

and heterogeneous sensors [Sal99] [Roc05]. The array of sensors is evolving [Gri03], increasing the complexity of the issue. To further complicate matters, these sensors communicate in an asynchronous fashion [Roc05] and the data that they produce is strongly time-dependent [Dem07]. This is primarily due to the nature of context data, which changes as frequently as the context it represents.

Like mobility, the problem with context-awareness can be reformulated as a problem of abstraction. In this case, it is a matter of taking raw data and abstracting it to a form that can be acted upon programmatically, and successfully hiding the nature of the sensors used to gather the information. This involves dealing with multiple data types, and writing programs that can act upon context information. Once these issues are addressed, applications can effectively use context to enhance the user experience.

1.4 Mobile Context Aware Computing

Mobile wireless devices open the door for context aware applications, yet impose challenges due to the limitations and variety of the devices. The field of Mobile Context Aware Computing (MCAC) deals primarily with the issues associated with software that can adapt to its context on distributed, heterogeneous devices. This includes adapting to the resource limitations and sensory capabilities of the devices themselves.

MCAC is an emerging field with many open issues. For instance, application designers in this sphere cannot treat the hardware as an isolated black box. MCAC

requires consideration of the limitations of the devices involved, the environment, and network connections.

1.5 Barriers to Entry

Development with MCAC is more difficult than traditional software development because of the use of context information, location-enhanced behaviors, and user mobility [Li04]. As discussed above, the devices themselves also impose difficulties to developers. The process and tools people use to design and test software will also differ a bit within MCAC, because special considerations for the target devices and the OS that they run must be made. Additionally, inputs to the program may not be generated by the user directly, but through the user's context, making unit testing and automation more complex. Together, these issues form a high barrier to entry, which is unfortunate for an emerging field that could benefit from experimentation, particularly by the research and academic communities.

Several attempts at lowering these barriers have already been made, ranging from toolkits to complete programming suites. Salber, et al., created the Context Toolkit to hide much of the specifics of interacting with low-level sensors in order to enable context-aware applications [Salb99]. It is based on the idea of the widgets, which abstract away the details of the context sensors to a more easily used form. This approach enables developers to create reusable solutions for low-level sensing mechanisms, then separately use those solutions to create applications. Grisworld, et

al., extended beyond the idea of the toolkit to explore the issues of extensibility and integration among context aware applications in ActiveCampus [Gris03a]. This system exemplifies the approach of creating distributed and connected context aware services, and explores the issues within this approach.

Du and Wang placed more focus on the development process [Du08]. They aimed at creating an entire suite for developers to work, including a development environment. In effect, his work is complementary to ActiveCampus; it addresses the human problem of how they perform their work, rather than the problems of how the technology works. Topiary addresses the human problem as well, targeting application designers rather than software engineers [Li04]. It attempts to allow experimentation with different interactions during early stage design, and to allow rapid iteration utilizing user feedback.

In order to quickly progress the MCAC field, both human and technology problems need to be addressed. Sensors need to be abstracted and applications need to integrate together. Designers need to easily design applications, and engineers need to implement them. Students and researchers need to address their burning research questions, without technology creating further difficulties. Although many efforts have been made to address these different issues, having one solution to experiment with these issues could potentially reveal subtle interactions and synergies between them. In complement to these efforts, our goal is to enable learning through experimentation more quickly and easily by supporting incremental development.

Furthermore, the solution should be easy to learn and extend, in order to avoid raising the barrier to entry.

1.6 Extensibility

The purpose of experimentation software is to enable learning through trial more quickly and easily than development of a full-scale solution. This way, results are generated and iterations take place faster and with less risk of wasted resources. In the domain of MCAC, it is particularly important that experimentation software support extensibility. "A typical system might regularly undergo the addition of new kinds of context sensors, modeled entity types, services, or end-user devices" [Gris03a]. Changes to applications and sensors may require changes to the context model [Roc05]. Experimentation software should be able to handle these changes quickly and easily.

Building a system with sufficient flexibility and generality introduces further issues. Often, a tradeoff must be made between flexibility and generality versus performance and scalability [Roc05] [Roc07]. The more versatile a system must be, the more work it must do to achieve that versatility. Also, one must be careful not to build software components for generality while losing utility. That is, sometimes when something is built to do anything, it sometimes it takes a lot of work to make it do something. Interchangeable pieces with defined interfaces are common practices

for handling change in a software system. However, some integration between parts is useful and beneficial from a performance standpoint [Gris03a].

In short, experimentation software, particularly when MCAC is concerned, must facilitate rapid change in the components of the system and the information that is passed between components.

1.7 Hypothesis

We hypothesize that many of the problems of mobile computing can be addressed by an extensible, dynamically reconfigurable context publish-subscribe architecture. A publish-subscribe architecture decouples producers and consumers of context information, insulating their communication from the low network speed and lack of robustness typical of mobile networks. Also, it naturally enables a system to evolve along with the sensing capabilities of mobile devices by incorporating new types of events for publication. Furthermore, dynamic reconfiguration allows devices to offload computation onto the network, compensating for the limited software that runs on small, battery-powered mobile devices. Extensibility in the software enables it to evolve along with the sensing capabilities of mobile devices by incorporating new data types. However, these capabilities alone are not enough to support MCAC. The addition of dynamic reconfiguration allows devices to introduce new computational functions at low effort, move computations close to their data source, or offload them onto the network to, say, compensate for the limited resources of small, battery-

powered mobile devices. In aggregate, these features comprise a flexible and easy-to-use experimentation platform for mobile devices.

1.8 Approach

We decided to test our hypothesis by creating UbiBot, a system of client and server software based on an extensible, dynamically reconfigurable context publish-subscribe architecture. The client software runs on mobile devices with weak connections to the network, while the server software is designed to run on stationary machines with strong network connections.

An experimenter who wants to create a new service need only create a server using the libraries included with UbiBot. The libraries provide the publish-subscribe communication architecture, and the means for clients to perform dynamic reconfiguration. The libraries are designed to be extensible with new data types.

We created a few diverse services on the UbiBot platform to determine if it is a system that experimenters could use to quickly and easily try out new services in the MCAC field. These services will show not only the range of software supported by UbiBot, but the effort involved in creating services with it.

1.9 Results

We were able to demonstrate the capability of UbiBot by creating a diverse array of applications. Furthermore, we showed the ease with which others could use the software to create a new MCAC application. Together, UbiBot's core mechanisms of subscription, hosting, and delegation lowered the barrier to entry for MCAC.

1.10 Structure of the Thesis

The remainder of this thesis is organized as follows. Chapter 2 describes UbiBot's client and service software. We focus on how the software simplifies the process of creating services. Chapter 3 discusses the communication between the parts, particularly on how the tenuous connections to the network are handled. Chapter 4 is an evaluation of the system, using a few representative services as examples. Chapter 5 provides a conclusion and discusses future work.

Chapter 1, in part, is a reprint of the material as it appears in "UbiBot – Prototyping Infrastructure for Mobile Context-Aware Computing" in *Proceedings of the Second Workshop on Pervasive Computing Education (PerEd '09)*. Erwin Vedar, W. Brian Evans, William G. Griswold, 2009. The thesis author was the primary investigator and author of this paper

CHAPTER 2: UBIBOT

MCAC is a challenging field because of the variety of devices, the common features of which being their difficulties: small size, weak connectivity, and lack of computing power. An effective programming infrastructure for this field would provide the building blocks to create a variety of interesting applications, and make use of the opportunities provided by context-awareness and mobility. It would further provide a communication protocol to connect those building blocks together. Also, it would be easy to learn and use, so that students and researchers can spend their time and efforts in conducting their experiments, rather than learning and setting up the infrastructure.

This section describes how UbiBot provides such an infrastructure by examining a few of its key mechanisms and features. UbiBot's core feature is the service. A service is a semi-autonomous computational unit that has the capability to publish events of a prescribed type, as well as subscribe to events of a type as prescribed by another service. A full-blown application consists of one or more services. For example, a mobile phone application may provide services for location, sound capture, image capture, etc. When an application consists of just a single service, it may be referred to simply as a service itself.

In order to tie the pieces together, we'll discuss how location-based reminders, one of the applications from ActiveCampus [Gris03a], could be implemented in UbiBot. Location-Based reminders can be thought of as an alarm clock that goes off

at a certain spatial occurrence, rather than a temporal one, as in a traditional alarm clock. That is, if one desires to be reminded to do something at a certain place, rather than a certain time, it would be useful to be reminded of the task when in that place. For example, in some cases, the best time to be reminded to buy cereal is when you're at or near the grocery store.

Specifically, Location-based reminders are signals sent to the user when certain criteria of his location are satisfied. For simplicity, the user can define the criteria via the GPS coordinates of a bounding box. When the user's coordinates fall within the bounding box, the alert is triggered. Thus the bounding box can be set around a building, so that when the user is near the building, he can reminded of a task that needs to be done in that building.

2.1 Subscription

In order to create context-aware applications that are distributed over the network, information must flow from the context-gathering sensors to the applications that will process and transform that information into something useful. On a resource-rich server, desktop, or laptop, with plenty of computing cycles, memory, and a strong network connection, this would not be a problem. However, with mobile devices that lack those luxuries, the dual burden of gathering the information and doing something interesting with it can be too much load for it to handle. This load is even greater for applications that combine and process information from several devices, such as in a

social-based application. In order to enable separation between context gathering and context processing, a communication protocol needs to support information flow from the sensors to the processors. In classroom and research settings with specific focus on a topic and tight deadlines, it's useful to use a simple protocol with a low learning curve. Thus the focus can remain on the topic of interest and not learning a complex information dissemination protocol.

The publish-subscribe paradigm is a simple approach that meets the need for distributing context information. It also provides several other benefits relevant to the mobile space. A device serves as a publisher of its context information, and other parties interested in that information, possibly to combine it with the information from others, are its subscribers. A publisher can have many subscribers; when it has an update, the information is sent to all of them. Conversely, a service can subscribe to the context information from several devices. For example, an application requiring the GPS location of several devices would subscribe to the GPS location of each of those devices. When one of them had an update to their location, they would publish that update to all of its subscribers, including that application. The application can then take the appropriate action based on the new location.

The nature of the mobile, distributed sphere of MCAC is that there are many different kinds of devices, and that variety continues to increase with time. Publish-subscribe is useful in this respect because it decouples the publishers and subscribers. From the point of view of the publisher, a subscriber is just another party on its list of subscribers interested in a particular type of information. Transactions are based

simply on the data that is provided, without any need to deal with the specifics of the devices themselves. Thus the heterogeneity of the devices is not an issue, nor is their evolution. The devices and services are free to grow and change independent of each other, as long as the protocol for communication between them is maintained.

In the example of location-based reminders, it would be possible for a mobile device to host the service on its own. It could monitor its user's location, and compare it to a local list of locations that have reminders associated with them. However, moving to the service to a different machine has several advantages; these are detailed in the next section. Assuming a distributed application, then, the mobile device would only be in charge of gathering the location of the user, and sending the information to the Location-Based Reminders service. That information would be relevant to the Location-Based Reminders service on an ongoing basis; it would need to track the location of the user in case he or she entered into a zone that has a reminder associated with it. To ensure that it has this information, the service would subscribe to the location of the user by sending a subscription request to the device. Thus when a device obtains a new location for the user and publishes it, the Location-Based Reminders service would receive it.

Consider an example in which Alice and her officemate Bob are trying to keep their shared office stocked with enough office supplies and snacks, both of which are currently running low. Alice already has plans to stop by the grocery store and the office supply store later during the week. She just has to remember to get supplies for the office in each location. Bob starts a Location-Based Reminders service on the

server in their office, and preloads it with the locations and relevant reminders for the grocery store and office supply store. He gives the UbiBot identifier for the Location-Based Reminders service ("LBR" in the example) to Alice.

Alice's GPS-enabled phone runs UbiBot. Whenever the GPS has fixed a new location, it sends out an update to everyone that has subscribed to her GPS location. Essentially, her update consists of two name-value pairs: her name and GPS location.

<ENTITYUPDATE><ENTITYNAME>Alice<PROPERTIES>

<PROP>GPS<VALUE>32.875697,-117.240573

In this case, "32.875697,-117.240573" are Alice's GPS coordinates, since she is near the Biomedical Library at the University of California, San Diego. Currently she has no subscribers.

Given the name of Bob's instance of the Location-Based Reminders service, she instructs her UbiBot instance to identify itself to it in order to begin participating in its service. As part of the negotiation for the service, the service subscribes to Alice's location information in order to provide its service. It sends the message:

<ENTITYUPDATE><ENTITYNAME>LBR<PROPERTIES>

<PROP>Subscribe<VALUE>GPS

This indicates to Alice's device that it should send an update to Bob's server whenever it has fixed a new location.

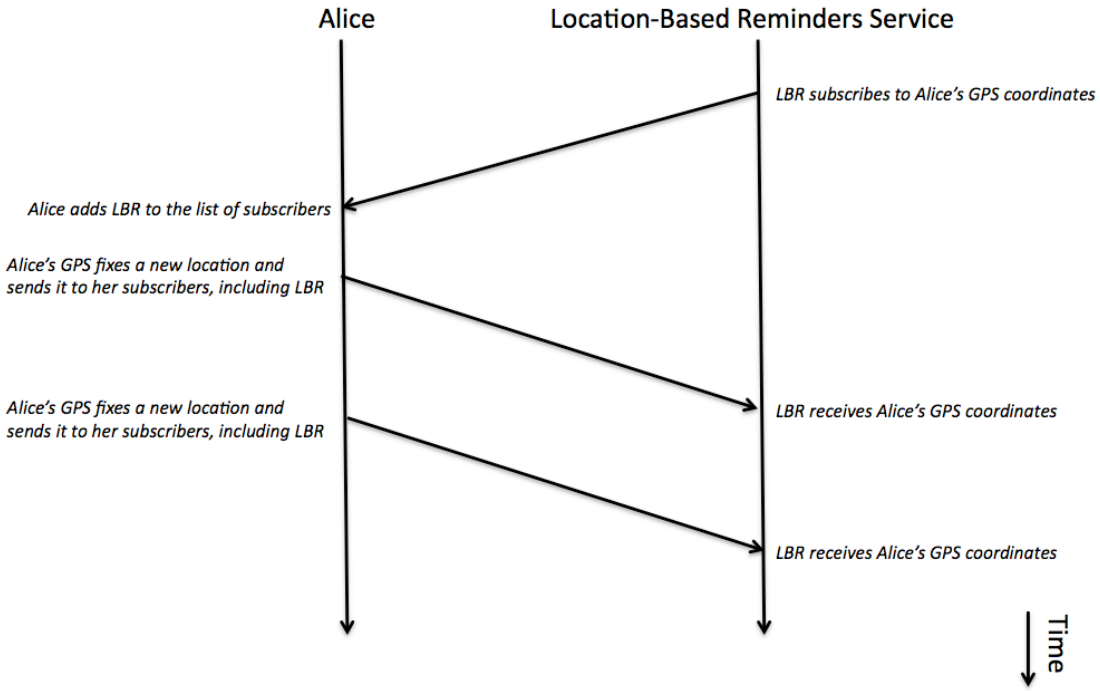

Figure 2-1: Location-Based Reminders Service subscribes to Alice's GPS

2.2 Hosting

Mobile devices are lower-powered and have less capacity than their desktop counterparts. However, there is still a demand for rich user experiences on these devices. One solution is to push all of the business logic to a resource-rich and stable host for the application, and simply handle the rendering of content on the remote device. This hosting approach is used by UbiBot to offload mobile devices while still providing rich experiences to users.

Since the focus of UbiBot is to enable experimentation with low overhead, interaction with services should be enable rich experiences, yet be easy to implement. This is accomplished through UbiBot facilities. Facilities are interaction widgets that appear on the device, but are populated and controlled by the host service. UbiBot features an extensible list of facilities that includes a web browser, buddy list, and text area. These can be used to render images, simple web pages, make a buddy list based on custom criteria, or display text to the user.

However, in the spirit of MCAC, it would be best not to assume the capabilities of the mobile device. Developers need to be prepared to deal with a variety of devices. What if the mobile device was simply a smart wristwatch, or had no screen at all? To this end, UbiBot further abstracts the facilities into kinds. A kind is a data type that can be handled by a facility.

Table 2-1: Currently supported kinds and typical facilities

Kind	Facility
Locatable	Map
Postable	Text Box
Renderable	Web Browser
Messageable	Instant Messenger

A combination of subscription and hosting is used to negotiate participation in a service by a device. The service can be agnostic about the actual capabilities of the device, as long as it supports the required kinds. Devices are free to evolve, change, and differ from each other, without concerns for compatibility with the existing services, as long as it can handle the necessary kinds.

In the example of Location-Based Reminders, the service would be hosted on a machine separate from the mobile device. It would receive updates through the subscription mechanism detailed in the previous section. In order to show the reminder to the user, the service could use the postable kind. On most devices, postable kinds would be displayed in text boxes. However, the reminders service doesn't control what facility the device uses. If UbiBot is running on an advanced wristwatch device with text-to-speech conversion, perhaps it would make sense for the device to handle the postable kind by reading the post aloud.

Negotiation of the kinds used in a service are handled when a UbiBot instance running on a device identifies itself to the service. When Alice instructs her device to identify itself to the service, it sends the following information to the bot:

<ENTITYUPDATE><ENTITYNAME>Alice<PROPERTIES>

<PROP>Supports<VALUE>Postable

<PROP>Supports<VALUE>Renderable

This tells the service that the user identified as Alice is using a device that can post text to the user (postable information) and also render simple web pages (renderable information). In Alice's case, it can open up a tab in the interface on Alice's device with a text box that accepts updates in order to accommodate the postable type, and/or a web browser to handle the renderable type. Bob's Location-Based Reminders service uses simple text reminders, so it only needs the postable kind. The service indicates to Alice's device that this is what it needs.

<ENTITYUPDATE><ENTITYNAME>LBR<PROPERTIES>
<PROP>Supports<VALUE>Postable

In turn, Alice's UbiBot instance opens a tab in the interface on her device with a text box that accepts updates. To enable the updates, Alice's UbiBot instance sends Bob's service a reference to the text box.

<ENTITYUPDATE><ENTITYNAME>Alice<PROPERTIES>
<PROP>Handle1<VALUE>Postable

The value appended to the handle property indicator is a unique numerical identifier (in this case, 1) that Alice's device uses to refer to the text box. When the Location-Based Reminders service wants to update the text in the box, it sends an update to Alice's device:

<ENTITYUPDATE><ENTITYNAME>Alice<PROPERTIES>

<PROP>Handle1<VALUE>Buy cookies

In order to be effective, though, these reminders should coincide with Alice's GPS location, so that she is reminded to buy cookies when she arrives at the grocery store, or printer paper when she is at the office supply store. Since Bob's service is subscribed to Alice's GPS location, it can do just that. It will receive Alice's location from her device when the GPS has fixed a new location. If that location is in or near the grocery store, the service sends an update to the text box on Alice's device reminding her to buy cookies. In the case of the office supply store, it can remind her to buy printer paper. Thus, she can be reminded at times that are relevant to her ability to take action, rather than at a scheduled time like a typical alarm clock. Thus she can go to each store at her leisure, and be reminded to get the right supplies at the right place. Furthermore, Bob could also get his own mobile device to participate in the service, so that he can also be reminded to resupply the office if he happens to be near the grocery or office supply stores.

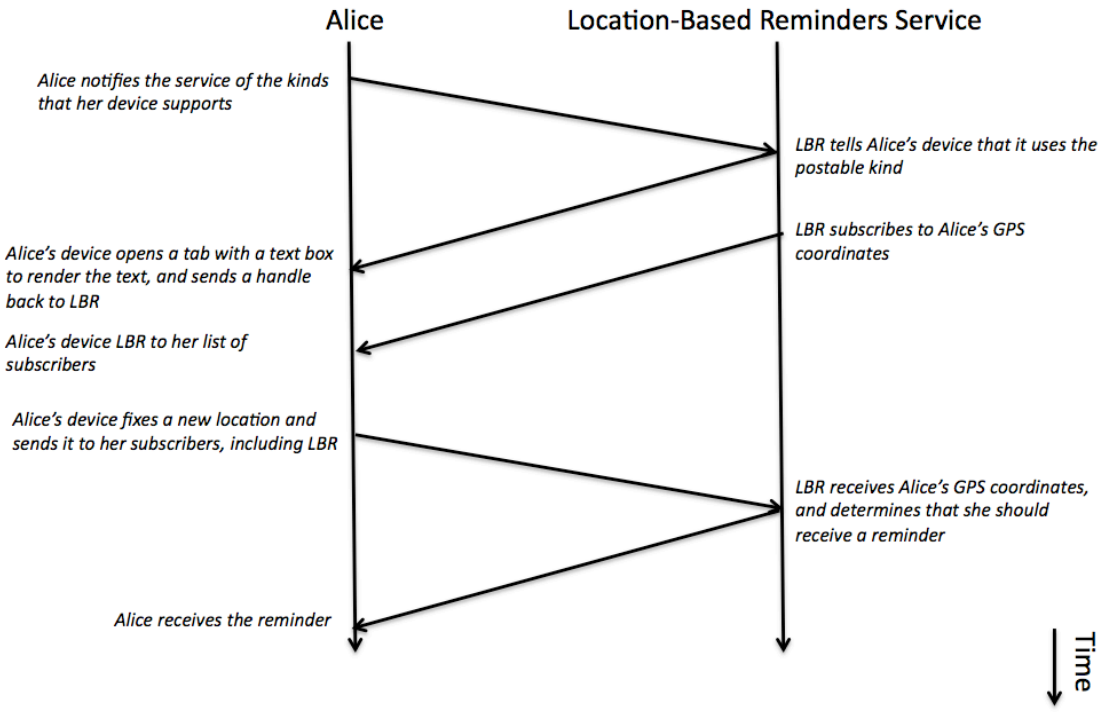

Figure 2-2: Location-Based Reminders negotiates kinds with Alice's device

2.3 Delegation

Sometimes a client may be able to provide a service to others, but with high power cost and low reliability, due to the frequent use of the unreliable network. An example is a client publishing its location to all subscribers. A more efficient solution would be for the client to publish its location once, and to have a non-mobile service store and distribute this information to other services. However, such a solution could incur substantial development delays during initial rapid prototyping. Ideally, the simple but ineffective location service could be developed initially, and then later the

more robust and efficient service could be added in later – at low cost – if desired.

In UbiBot, this is achieved via the proxying mechanism. Any context information that can be gathered by the device can be proxied by another machine. All existing and future subscriptions are forwarded to the proxy. When the device has an update to publish, it publishes it once to the proxy, which in turn fulfills all of the subscriptions. Thus the mobile device is relieved of the burden of publishing multiple copies of the same information.

Since mobile devices have inherently less stable network connections than standard stationary machines, it makes sense that such stationary machines serve as the proxies for mobile devices. In doing so, context information can flow more reliably to all of the subscribers, even if the mobile device suffers from intermittent disconnections. Furthermore, by design, UbiBot is dynamic. That is, new services and subscriptions are added at runtime. Also, as a user moves to different locations within one or more wireless networks, the network connection for his or her device will change. Consequently, then, the proxying mechanism itself is dynamic, both to cope with these issues and to facilitate experimentation with different arrangements of devices.

Consider the case in which several instances of the Location-Based Reminders service are running. One could imagine that each instance could be run by a different group of people, and an active student would be interested in the reminders from each of them. For example, one could be run by the student's research group, another by a social club, another by the student for his or her own personal reminders, and another

by his or her dorm. This arrangement allows each group to manage their own reminders to meet their needs, and ensures that there's no single point of failure for all of one person's reminders. However, each reminder service would need the location of the user's device, and publishing that location would involve sending that data several times. Imagine that the user's device is low on battery, and that its connection to the network isn't very strong, so perhaps a few of the publications get lost. The student could start a proxy service on a machine that sits in his or her dorm room, and use UbiBot's proxying mechanism to forward all of the subscriptions to that proxy. When publishing updates, the device need only successfully communicate once, to the proxy. The proxy then passes along the update to all of the subscribers. Thus the device has offloaded the work onto the proxy, and achieved more reliable delivery of information.

In summary, there are several benefits to proxying. Once the proxy has the client's information, the client device no longer has to compute latitude and longitude; it can turn off its GPS unit or bypass its GSM location translation facility, whichever method it was employing. It uses the network much less to communication its location, decreasing power consumption. If the client drops off of the network briefly, its location remains available to its subscribers.

Continuing Alice and Bob's example, suppose she continues to participate in the Location-Based Reminders service that Bob set up in her office. Furthermore, she participates in a Location-Based Instant Messaging service (LBIM), as well as a Location-Based Automated Tour Guide of her campus (LBAT). These bots are

described in Chapter 3. Each of these services relies on timely updates to Alice's location in order to function. However, each time she updates her GPS location, she sends out three copies of the same information, which can drain her device. Alice starts up a location proxy service from a server she runs at home. When it comes online, she has it set to identify itself as a proxy to her device.

<ENTITYUPDATE><ENTITYNAME>AliceProxy<PROPERTIES>
<PROP>Proxy<VALUE>GPS

Having received this introduction, Alice's device automatically records then forwards all current and future subscriptions to her GPS location on to the proxy.

<ENTITYUPDATE><ENTITYNAME>Alice<PROPERTIES>
<PROP>Subscriber<VALUE>LBIM
<PROP>Subscriber<VALUE>LBR
<PROP>Subscriber<VALUE>LBAT

Now, when Alice's device fixes a new GPS location, it only gets sent once by her device to the proxy, via the standard update method. The proxy sends an acknowledgement to Alice's device.

<ENTITYUPDATE><ENTITYNAME>AliceProxy<PROPERTIES>

<PROP>ACK<VALUE>GPS

If the proxy were to fail to send the acknowledgement, it would be an indication to Alice's device that it must revert to the old method of sending an update to each of its recorded subscribers. However, if Alice's proxy is running correctly, it can send the updates via a hard connection to the network to each of the subscribers.

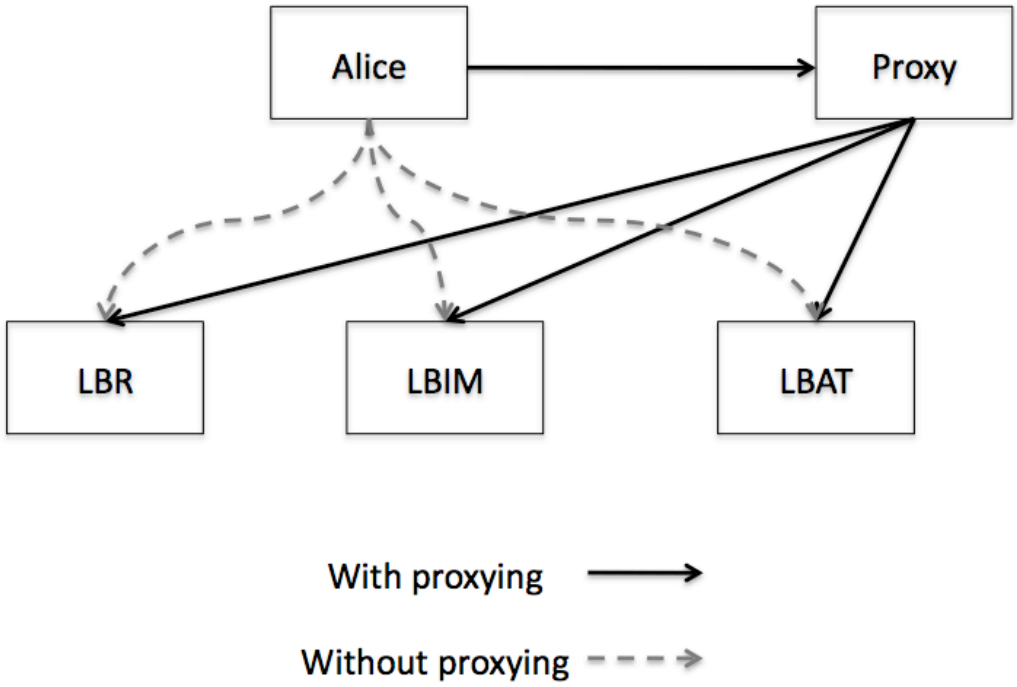

Figure 2-3: Flow of information with and without proxying

2.4 Summary

The Location-Based Reminders example showed how UbiBot can be used to create a context-aware application using subscription and hosting. The proxying mechanism enables the device to offload some of the work to another machine. What makes UbiBot unique is its ability to "program" the network through hosting and delegation, enabling developers to start with a very basic system, and then gradually evolve it into a more mature form.

Chapter 2, in part, is a reprint of the material as it appears in "UbiBot – Prototyping Infrastructure for Mobile Context-Aware Computing" in *Proceedings of the Second Workshop on Pervasive Computing Education (PerEd '09)*. Erwin Vedar, W. Brian Evans, William G. Griswold, 2009. The thesis author was the primary investigator and author of this paper.

CHAPTER 3: EXTENDING AND EXPERIMENTING WITH UBIBOT

A good programming infrastructure helps to bridge the gap between the programming interface exposed by the device and the developer's vision of a useful product or experiment. It handles the lower-level details of the interface so that the developer can pursue the higher-level results more quickly and with greater focus. Typically, infrastructures suggest a certain style of programming through the libraries they provide, but a good infrastructure still gives the developer the flexibility to create a variety of applications.

Infrastructures have great potential in experimental, research, and classroom settings because of the increased increase in development speed they promise. Academic endeavors often focus on a specific hypothesis or lesson for a short period of time; programming infrastructures are useful because they handle details that are irrelevant to the question at hand. In this way, the little time that students and researchers have to do their work can be spent pursuing results, not figuring out an API or setting up a framework.

We hypothesize that UbiBot is flexible and easy enough to use for experimentation in research and classroom settings. In order to show the usefulness and ease of experimentation with UbiBot, we developed a variety of services on the platform. These services show that UbiBot supports a range of applications.

3.1 Location-Based Instant Messaging

The basic criteria for an MCAC application is that it runs on a mobile device, and uses context to enhance the user experience. In order to show the sufficiency of UbiBot for such a minimal foray into the UbiBot realm, we developed a Location-Based Instant Messaging service on the platform. It is a good minimal example because it leverages both mobility and context to enhance online communication between users.

Griswold, et al., developed a location-based instant messaging system on the ActiveCampus platform [Gris03b]. The general idea is to take the concept of instant messaging, and augmenting the availability status of participants with location information. That is, when a user is deciding whether to engage a buddy in conversation, that user can know a bit more about their buddy's current status beyond their being online or offline. The user will have some concept of the buddy's location. As noted in the ActiveCampus project, users are more likely to message buddies in close proximity than buddies that are further away [Gris03b].

Given the belief that the more proximate buddies are more interesting, one possible interface design would be to only show buddies that are nearby. This way, only the most relevant and desired options are shown to the user. UbiBot Location-Based Instant Messaging service takes this approach and populates a buddy list on the user's device based on a configurable distance threshold between the two parties. If they are within a certain distance, they appear on each other's buddy lists. If not, they do not appear on each other's buddy lists.

There are several tasks necessary to provide this service. The locations of each participant need to be gathered, their pair-wise distances calculated, and buddy lists updated accordingly. To minimize the demands on the mobile devices that participate in the service, it is reasonable to off-load the location aggregation, distance calculations, and buddy list management to a service at a fixed location. The UbiBot Location-Based Instant Messaging service achieves this through the hosting mechanism described in Chapter 2. A central server is set up that receives a each participant's location through the subscription mechanism, calculates their respective pair-wise distance, then updates the buddy lists on the devices accordingly. The central computer, then, alleviates the devices of any work aside from what is strictly necessary, i.e. gathering and sending on location information, displaying a buddy list to the user, and sending and receiving instant messages.

All of this functionality on the device side is built into the client. That is, no changes to the client device are necessary to participate in the Location-Based Instant Messaging service. UbiBot's client software can gather location information, and the built-in subscription mechanism forwards that information on to interested parties. In the case of Location-Based Instant Messaging, the service filters and combines the information before sending it out to the interested parties. The service then uses UbiBot's buddy list to display the buddies nearby; the buddy list feature is a list that can be populated and manipulated by outside services.

UbiBot is built on top of instant messaging (IM) as its communication substrate. That is, services and clients send instant messages to each other in order to

communicate via Microsoft Messenger. Instant messaging provides location-independent naming of services, asynchronous messaging, and routing through firewalls. This makes it relatively easy to set up a basic MCAC system from scratch. All that is needed is the creation of new IM accounts for each of the relevant nodes in the system (participating fixed computers or mobile devices). It is not necessary to set up a centralized server or related software. Nodes can host one or more services. A message processor watching the IM channel parses incoming messages (event objects) to recognize the destination service and route appropriately. Messages between services on the same node are handled with internal routing, avoiding the IM channel.

Before discussing the development of the service software, we examine the interaction between the service software and the client. As described in Chapter 2, the user enrolls in the service by introducing itself to the service and telling it what facilities it supports.

<ENTITYUPDATE><ENTITYNAME>LBIMUser<PROPERTIES>
<PROP>Supports<VALUE>Mesageable

The service receives the message, and selects the Messageable facility so that it can receive a handle to a buddy list on the device.

<ENTITYUPDATE><ENTITYNAME>LBIM<PROPERTIES>
<PROP>Supports<VALUE>Messageable

This instructs the client's device to open a tab in its interface with an empty buddy list. This buddy list will be used to show a subset of buddies, chosen by the service, based on the buddy's proximity to the user. In order to carry out these tasks, the service subscribes to the client's GPS location.

<ENTITYUPDATE><ENTITYNAME>LBIM Bot<PROPERTIES>
<PROP>Subscribe<VALUE>GPS

Whenever there is an update to the client's (Bob's) location, it sends it to the service.

<ENTITYUPDATE><ENTITYNAME>Bob<PROPERTIES>
*<PROP>GPS<VALUE>*32.881800, -117.233575

Assuming that at least one other buddy has carried out the same process with the service, location-based instant messaging can begin. Given the location of two entities, the service calculates the distance between the two. With that information, it can decide if they should appear on each other's buddy lists due to their proximity.

Clearly, the buddy list can be populated based on location, but other uses of context information are possible. This service demonstrates the ability of UbiBot to support location-based (and thus similar context-aware) applications.

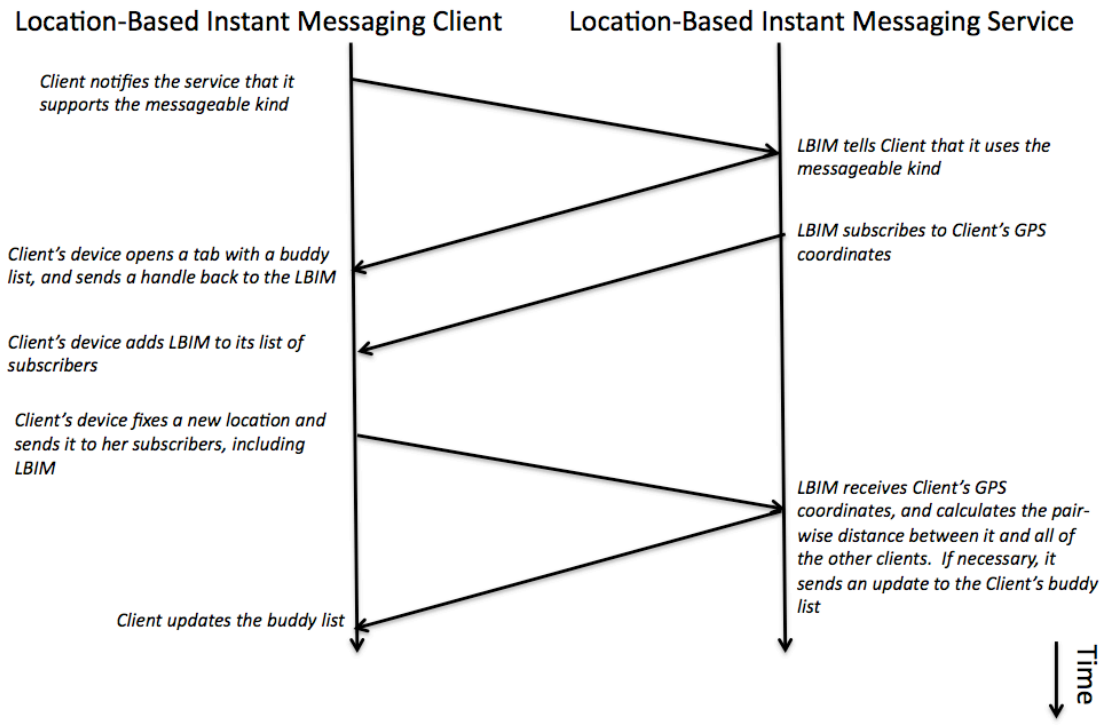

Figure 3-1: Client-service communication in Location-Based Instant Messaging

3.2 GPS Proxy

One of UbiBot's design philosophies is to perform work where it makes most sense. Mobile devices travel through interesting contexts, and feature an expanding array of sensors such as microphones, GPS, and cameras. However, compared to their more stationary counterparts on the desktop, they are lower-powered and weakly connected to the network. Thus, it makes sense that these devices gather context information, but as much as possible the aggregation and manipulation of the data, particularly if it is drawn from several mobile devices, be handled by a server at a

fixed location. Furthermore, since the server has a more dependable connection to the network, it can serve as a central point of communication for the more weakly-connected devices. This way, it can serve as a single source of truth providing the most up-to-date information, should devices become partitioned from one another.

Generally, the UbiBot mechanism for this is a form of proxying, as described in Chapter 2. This allows a device to choose another machine, presumably one with greater computation power and a stronger connection to the network, to handle requests for context information. That is, requests for context information about the user will be handled by a machine better able to handle it. As long as the device keeps the proxy up to date, the proxy will provide the correct context information. If the device becomes partitioned from the network and cannot provide updates to the proxy, the proxy can still report the most up-to-date information that has been received. Then any services in which it was participating can still include the partitioned device, if it still makes sense to do so. If the proxy fails, the device can revert back to its original mode of servicing requests on its own.

To demonstrate the proxying capabilities of UbiBot, we present GPSBot, a service designed to serve as a proxy for GPS information. GPS information is useful for any application that uses location to enhance its functionality. However, the varying location of the device (and hence its interesting GPS coordinates) implies that the strength of network connection may vary due to the changing distance from the wireless router. Thus the ability of the user to participate in location-aware applications may be hampered. GPSBot addressed this problem by offloading the

distribution of location information to a stronger machine with a more reliable connection to the network. If the device has a spotty connection to the network, its last known location can be maintained by the GPSBot until the device reconnects and is able to send another update.

The device is also spared the additional load of sending out redundant information. For example, if the user participates in *n* different location-aware application simultaneously, a single update to location will be sent *n* times by the device. However, this is redundant and the cost to the device in terms of computation cycles and battery life may negate the usefulness of the applications. GPSBot addresses this by allowing the device to send a single update with confidence that it will propagate the new location information to all interested parties.

In research and classroom settings, dealing with real-world network connection issues may impede the progress toward proving or disproving the learning hypothesis. By hiding the spottiness of the device's network connection from location-aware experiments, this obstacle no longer poses as much of a problem to the researcher or student. This way, the changing context of the device can be explored without interference from network connectivity issues.

Let's examine how the GPSBot functions and alleviates the user's device of communication issues. Consider Alice, who currently participates in the Location-Based Reminders service described above. In order to serve her requests, the LBR service needs to know her location, and so it sends a subscription request to her.

<ENTITYUPDATE><ENTITYNAME>LBR<PROPERTIES>

<PROP>Subscribe<VALUE>GPS

Thus when Alice's location changes, for example, to 32.875697, -117.240573, her device knows to publish the update to LBR, a subscriber.

<ENTITYUPDATE><ENTITYNAME>Alice<PROPERTIES>

*<PROP>GPS<VALUE>*32.875697,-117.240573

Now Alice also wishes to participate in Location-Based Instant Messaging so that she can communicate to fellow students near her. Of course, the LBIM service needs to know her location as well, so it sends a subscription request to Alice as well. Now, when Alice's location changes, it sends two identical updates: one to LBR and one to LBIM. To reduce the redundant work her mobile device has to perform, she calls Bob back at the office and asks him to activate GPSBot. Once online, GPSBot sends Alice's device a notice that it can proxy her GPS information.

<ENTITYUPDATE><ENTITYNAME>GPSPxoy<PROPERTIES>

<PROP>Proxy<VALUE>GPS

For simplicity, UbiBot automatically accepts any proxying requests. To set up, Alice's device tells GPSBot that she currently has two subscribers: LBR and LBIM.

<ENTITYUPDATE><ENTITYNAME>Alice<PROPERTIES>

<PROP>Subscriber<VALUE>LBIM

<PROP>Subscriber<VALUE>LBR

Also, it notifies LBR and LBIM that GPSBot will be the proxy. When Alice has a location update to publish, rather than send two redundant updates, it sends a single unique update to GPSBot.

<ENTITYUPDATE><ENTITYNAME>Alice<PROPERTIES>

*<PROP>GPS<VALUE>*32.875697,-117.240573

GPSBot then forwards this update to LBR and LBIM, fulfilling their subscription requests. Also, GPSBot echoes back the request to Alice's device as an acknowledgement that the request was received.

After this setup, Alice decides to participate in yet another location-aware service, the Location-Based Automated Tour Guide, which will be described in the next section. As the name implies, it too needs to know Alice's GPS coordinates. However, this time when Alice's device receives the request, it immediately responds that the GPSBot will be the proxy for her location information. Alice's device sends the new subscription on to GPSBot, which will now fulfill the requests when Alice's device publishes a new location. However, now that she is a client of GPSBot, her

device only sends a single update do GPSBot, rather than to every one of her subscribers.

If the customer were to also participate in the Location-Based Automated Tour Guide, as described in the next section, a new subscription request would be sent to Alice's device. Alice's device would immediately forward the request to the proxy.

As with any proxying relationship, if the proxy should fail, the default behavior is for the user's device to resume sending out her own updates. This condition is detected when the GPSBot fails to acknowledge an update. Since the initial subscription requests are recorded by and fulfilled through the user's device, the device has a record of all of its current subscribers. Thus, the device can take over update duties without any loss of interaction with the applications involved.

3.3 Location-Based Automated Tour Guide

Mobile devices continue to evolve their abilities to render multimedia content through native applications such as their web browsers. Combined with the location-sensing abilities of mobile devices, this presents an opportunity for rich user experiences that combine context with media. The Location-Based Automated Tour Guide service showcases UbiBot's capability in combining context with media.

LBAT takes the concept of Location-Based Reminders a step further. Recall that LBR posts a text reminder created by the user when the user meets certain location criteria. LBAT, on the other hand, utilizes the device's web browser to render

content for the user. Thus the "reminder" can be in any media form supported by the user's browser. The intent of the "reminder" though, is not to be an item of the user's personal to-do list. It is instead a link created by whoever set up the LBAT, for anyone interested in information about their current location, similar to a tour guide. For instance, if the user were running the service 32.881800, -117.233575, which is the location of the Computer Science and Engineering (CSE) department of the University of California, San Diego, he or she could be served with the department's home page. One can imagine that a special page might be created for anyone taking a campus tour.

The setup of LBAT is similar to that of the LBR service. First Alice sends a request to the LBAT, introducing herself and her device's capabilities.

<ENTITYUPDATE><ENTITYNAME>Alice<PROPERTIES>
<PROP>Supports<VALUE>Postable
<PROP>Supports<VALUE>Renderable

The LBAT responds to Alice by requesting a handle to Alice's renderable facility.

<ENTITYUPDATE><ENTITYNAME>Bob<PROPERTIES>
<PROP>Supports<VALUE>Renderable

Alice's device sends back the handle.

<ENTITYUPDATE><ENTITYNAME>Alice<PROPERTIES>

<PROP>Handle1<VALUE>Renderable

Then the LBAT subscribes to Alice's GPS information as described in Section 2.1. If Alice is currently using a proxy, this request is forwarded to her proxy and the LBAT is notified. Now, whenever Alice's device (or her proxy) sends out updates, the LBAT receives the update and matches her location to a webpage, if any exist for her location. If one does exist, it sends Alice a message.

<ENTITYUPDATE><ENTITYNAME>LBAT<PROPERTIES>

<PROP>Handle1<VALUE>http://www.cs.ucsd.edu/

This indicates to Alice's UbiBot application to open a new web browser tab at this location. In the case of the Alice being near the CSE building, the department website (*http://www.cs.ucsd.edu/*) would be appropriate. She would see information, perhaps a mixture of text and pictures, pertaining to the department.

In order to support multimedia content, UbiBot makes use of the multimedia capabilities of the device's web browser. This way, it can leverage any new developments in the browser; advances in the browser are advances in the capabilities

of UbiBot. LBAT demonstrates how UbiBot can be used to combine the multimedia capabilities of the device with context information.

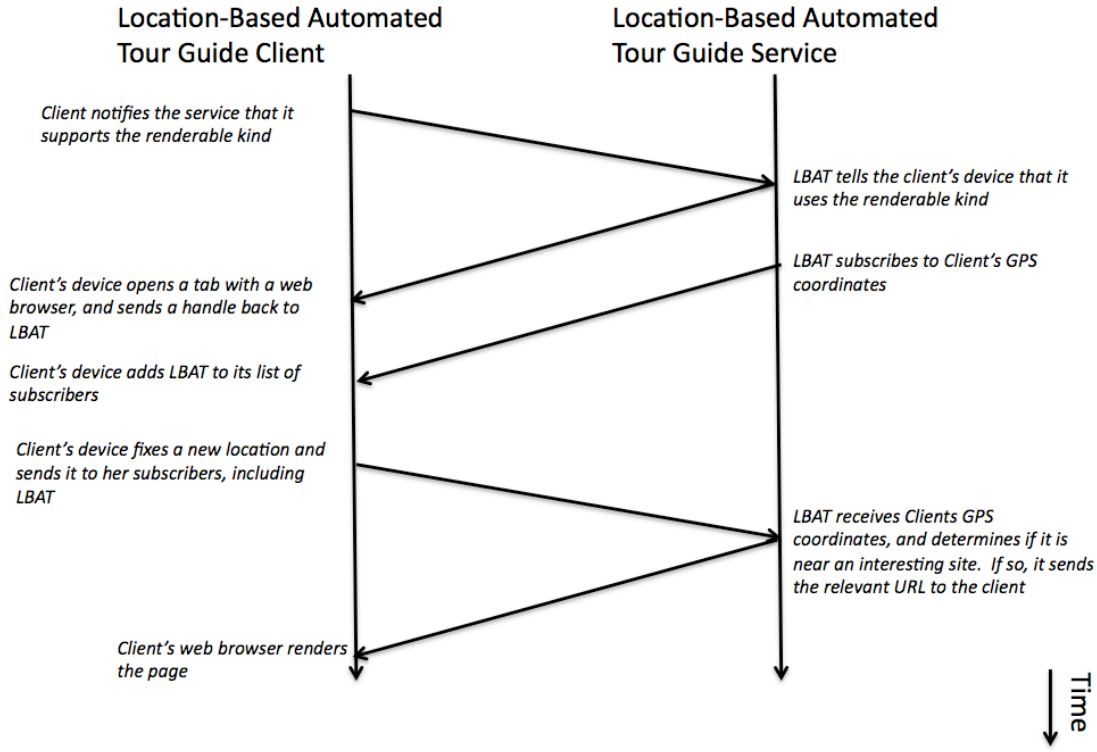

Figure 3-2: Client-service communication in Location-Based Automated Tour Guide

3.4 Enhanced Location-Based Reminders Service

As a preliminary evaluation of the UbiBot concept, two students taking a project class were recruited to develop a location-based reminder service for Windows Mobile phones (HP 6945's). As a twist on the usual reminder service, a reminder

could be sent to a buddy, not just oneself. Reminders could be limited to being delivered within a given time range, and could be reset for later delivery if delivered at an inconvenient time.

The students divided their application into server and client components. The client was developed as a plugin that could be hosted by a hosting service on a phone. The hosting service provided a skeleton GUI that permitted a "tab" to be plugged in to its display. The pre-existing client framework also consisted of services that published the phone's GPS location and GSM observations.

At a high level, an instance of the application is set-up by the phone's client hosting service, by sending subscribe reminder to the location service (this may be initially typed in by the phone's user or read from a simple scripting file). The service sends back a handle to the plug-in, which the client loads. The service also subscribes to the phone's location. (Alternatively, the loaded plug-in could have subscribed to the phone's location, allowing it to filter unneeded location events, when the phone isn't moving significantly.)

The students were advised to take an incremental development approach, starting with the most basic self-reminder application, later adding time windows for reminders, and then the ability to remind buddies. For this latter "fancy" feature, its core was simply to subscribe a chosen buddy to a specific reminder, rather than one self. The last incremental enhancement to the application was to delegate the location, as in the previous example. With a single statement of code, the students delegated the GPS location calculation to a GSM {latitude,longitude} UbiBot service that had been

developed earlier on top of Google's hidden API for its "My Location" function of Google Mobile Maps.

The server component consists of five classes, comprising 469 non-blank non-comment lines of code. The client component consists of four classes, comprising 551 lines of code (excluding GUI code generated by Visual Studio), for a total of 1020 LOC. About half of the client code is GUI code, and the students reported that most of their time was spent on getting the GUI to work properly, which required the use of C# delegates, a concept unfamiliar to them as Java programmers. Still, the students had little trouble completing the project in the 10-week class term, and most of their effort was concentrated into a few of weeks scattered across the quarter.

The ability to employ incremental development, especially in adding the "fancy" features of the application (reminders to buddies and delegated location calculation), allowed the professor (the third author) to define concrete milestones throughout the quarter that could be verified with a running demonstration, avoiding ugly surprises and saving all but the last feature from the (unfulfilled) prospect of failure.

Chapter 3, in part, is a reprint of the material as it appears in "UbiBot – Prototyping Infrastructure for Mobile Context-Aware Computing" in *Proceedings of the Second Workshop on Pervasive Computing Education (PerEd '09)*. Erwin Vedar, W. Brian Evans, William G. Griswold, 2009. The thesis author was the primary investigator and author of this paper.

CHAPTER 4: DISCUSSION

4.1 Enabling MCAC

The Location-Based Automated Tour Guide and Location-Based Instant Messaging demonstrate that UbiBot supports mobile context-aware applications. However, as mentioned above, it is necessary that UbiBot provide more than mere functionality in order to be considered a true enabler for MCAC. It must address the limitations of mobile devices and wireless networks.

Certainly, the bots demonstrate the use of context-awareness. Both use the location of the mobile device to enhance the user experience. Furthermore, UbiBot addresses issues associated with mobility: tenuous connections to the network as well as underpowered/under-supported devices. For both issues, UbiBot's design philosophy is to offload the burden from the mobile device onto another device (usually a desktop machine) that can handle it more easily.

GPSBot exemplifies UbiBot's approach to connectivity issues for mobile devices. Rather than forcing the mobile device to manage the demands of subscribers to its context, the GPSBot (and other bots using the proxying mechanism) alleviates that strain from the device by acting as a forwarding service for the device. Furthermore, assuming that the service is provided on a machine with a sturdy connection to the network, the GPSBot can continue to fulfill subscriptions even when the mobile device using its services repeatedly connects and disconnects from the

network. From the point of view of the subscribers, the device of interest has the network connection of a desktop machine, with the interesting contextual information of a mobile device.

UbiBot's approach to underpowered devices is similar to that used to address connectivity issues: if the device can't handle the burden, shift it to a machine that can. The mobile device's proximity to the user is the primary benefit it provides versus the desktop machine. Being attached to the user, it is the gatherer of information as well as the user's interface. Behind the scenes, the system architect has the option of hosting the service on another machine, thus alleviating the device of computation that is not strictly necessary to do locally. The choice between computing locally on the device or remotely on another machine can be made based on the results of experiments, as the answer will vary from service to service and from device to device.

4.2 Enabling Experimentation

As shown in Section 3.4, with little effort UbiBot was built upon for a project class without the need to write much code. It did so by supporting incremental development while imposing a low learning curve.

Developing for UbiBot requires very little prior knowledge of network or web protocols. Communication takes place over an existing instant messaging platform, MSN Messenger, with which many students will already be familiar. This also makes

intercepting and debugging communications much easier, and it can be done from any device that can send and receive messages over the IM network. The format of communication is a simplified XML-like format.

In terms of performance, one concern might be over the performance of Instant Messaging as a communication substrate. By timing the round trip times between mobile devices and a laptop, we determined that, over a typical home wireless network, instant messages typically take less than 2 seconds round-trip. More precise measurements are outside the scope of the current paper. As long as context information or communication doesn't need to be faster than this (which may be the case in prototyping/experimenting environments not focused on performance), this time should be acceptable.

CHAPTER 5: CONCLUSION

5.1 Summary

Mobile Context-Aware Computing is a field that presents both expanded possibilities for rich applications, and high barriers to developing those applications. In order to quickly and easily experiment in the field, we created UbiBot. UbiBot is a simple infrastructure that can be built upon and expanded. It provides the basic means to set up and communicate between mobile clients and services, gather and send context information, as well as the means to create simple UI experiences.

5.2 Contributions of the Project

The following is a summary of the primary contributions of the thesis.

A method for offloading the management of context. UbiBot allows the dynamic restructuring of the flow of context information to increase robustness of the network configuration. Developers can set up a client and server in one network configuration, run experiments, and then quickly change the configuration to include any number of proxies.

A library for developing context-aware applications on mobile devices. The UbiBot library enables experimenters to create context-aware multimedia applications on mobile devices. The UbiBot client software runs on the devices to allow the developers to take advantage of the context-gathering capabilities of the device, as well as display graphics and other feedback to the users. The software was written to be extensible, to allow for the incorporation of new types of sensors and display capabilities. The server software is designed to allow the user to create a simple server that processes UbiBot messages, without the steep learning curve or limitations of creating a standard web server. Both are based on the C#.Net platform, meaning that it can be adapted to any device that supports the .NET framework.

Sample context-aware mobile applications. Oftentimes, the best way to educate users quickly on how to use a new framework is to provide examples. It not only shows how the it is used, but also the style in which to use it. We developed a couple of sample applications in the exploration of UbiBot in order to show the range of capabilities of the software, which can also be used as examples for future developers. The Location-Based Instant Messenger provides a sample for users who want to develop mobile applications that create custom buddy lists based on context. The Location-Based Automated Tour Guide illustrates how to use context to display content to the users beyond the text format.

5.3 Future work

UbiBot is designed to embrace a variety of devices, with various sensors and facilities. It is currently targeted for the HP iPAQ hw6945 mobile phone device, but we would like to bring UbiBot to a wider variety of devices in order to reach a broader academic audience. Porting UbiBot to even more dramatically different form factors, such as a wristwatch-like device would allow us to explore the limits of the UbiBot framework, and different modes of interaction.

Since we aimed to create an easy way for UbiBot servers to be created, it would be interesting to compare it against another facility for convenient server creation: cloud computing. By creating a cloud-compatible version of UbiBot, it would be possible to explore the implications that cloud computing could have on the classroom and the study of mobile devices.

While expanding the scope and reach of UbiBot, we'd also like to focus on its purpose: to enable rapid experimentation in the classroom and in research. The true test of UbiBot would be to take UbiBot back to the classroom and laboratory and see what applications students and researchers are able to develop.

REFERENCES

[Ance02] Emmanuelle Anceaume, Ajoy K. Datta, Maria Gradinariu, and Gwendal Simon. Publish/Subscribe Scheme for Mobile Networks. In *Proceedings of the second ACM international workshop on Principles of mobile computing (POMC '02)*, pages 74-81, October 2002.

[Capo03] Mauro Caporuscio, Antonio Carzaniga, and Alexander L. Wolf. Design and Evaluation of a Support Service for Mobile, Wireless Publish/Subscribe Applications. *IEEE Transactions on Software Engineering*, 29(12):1059-1071, December 2003.

[Davi04] Oleg Davidyuk, Jukka Riekki, Ville-Mikko Rautio, and Junzhao Sun. Context-Aware Middleware for Mobile Multimedia Applications. In *Proceedings of the 3rd international conference on Mobile and ubiquitous multimedia (MUM 2004)*, pages 213-220, October 2004.

[Demi07] Thanos Demiris. Context Revisited: A brief survey of research in context aware multimedia systems. In *Proceedings of the 3rd international conference on Mobile multimedia communications (Mobimedia '07)*, 2007.

[Deva07] Anusiriya Devaraju, Simon Hoh, and Michael Hartley. A Context Gathering Framework for Context-Aware Mobile Solutions. In *Proceedings of the 4th international conference on mobile technology, applications, and systems and the 1st international symposium on Computer human interaction in mobile technology (Mobility '07)*, pages 39-46, 2007.

[Dey01] Anind K. Dey. Understanding and Using Context. In *Personal and Ubiquitous Computing*, 5(1):4-7, 2001.

[Du08] Weichang Du and Lei Wang. Context-Aware Application Programming for Mobile Devices. In *Proceedings of the 2008 C^3S^2E conference (C^3S^2E '08)*, pages 215-227, 2008.

[Faro04] Umar Farooq, Shikharesh Majumdar, and Eric W. Parsons. Engineering Mobile Wireless Publish/Subscribe Systems for High Performance. In *Proceedings of the The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS '04)*, pages 295-305, 2004.

[Gadd08] Abdulbaset Gaddah and Thomas Kunz. A Pro-active Mobility Extension for Pub/Sub Systems. In *Proceedings of the 1st international conference on MOBILe*

Wireless MiddleWARE, Operating Systems, and Applications (MOBILWARE '08), February 2008.

[Gris03a] William G. Griswold, Robert Boyer, Steven W. Brown, and Tan Minh Truong. A Component Architecture for an Extensible, Highly Integrated Context-Aware Computing Infrastructure. In *Proceedings of the 25th International Conference on Software Engineering*, pages 363-372, May 2003.

[Gris03b] William G. Griswold, Patricia Shanahan, Steven W. Brown, Robert Boyer, Matt Ratto, R. Benjamin Shapiro, Tan Minh Truong. ActiveCampus—Experiments in Community-Oriented Ubiquitous Computing. In *IEEE Computer*, 37:73-81, 2003.

[Li04] Yang Li, Jason I. Hong, and James A. Landay. Topiary: A Tool for Prototyping Location-Enhanced Applications. In *Proceedings of the 17th annual ACM symposium on User interface software and technology (UIST '04),* pages 217-226, October 2004.

[Muhl04] Gero Muhl, Andreas Ulbrich, Klaus Hemann, and Torben Weis. Disseminating Information to Mobile Clients Using Publish-Subscribe. *Internet Computing*, 8(3):46-53, May-June 2004.

[Roch05] Ricardo Couto A. da Rocha, Markus Endler. Evolutionary and Efficient Context Management in Heterogeneous Environments. In *Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing (MPAC 2005)*, pages 1-7, November-December 2005.

[Roch07] Ricardo Couto A. da Rocha, Markus Endler. Domain-based Context Management for Dynamic and Evolutionary Environments. In *Proceedings of the 4th on Middleware doctoral symposium (MDS '07),* November 2007.

[Salb99] Daniel Salber, Anind K. Dey, and Gregory D. Abowd. The Context Toolkit: Aiding the Development of Context-Enabled Applications. In *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit (CHI '99)*, pages 434-441, 1999.

[Samu01] Michael Samulowitz, Florian Michachelles, Claudia Linnhoff-Popien. Adaptive Interaction for Enabling Pervasive Services. In *Proceedings of the 2nd ACM international workshop on Data engineering for wireless and mobile access (MobiDE 2001)*, pages 20-26, 2001.