System Support for Rapid Ubiquitous Computing Application Development and Evaluation

Manuel Roman[1], Jalal Al-Muhtadi[2*], Brian Ziebart[2*]
Roy Campbell[2*], M. Dennis Mickunas[2*]

[1]DoCoMo Labs, USA
roman@docomolabs-usa.com
[2]Department of Computer Science, University of Illinois at Urbana-Champaign
{almuhtad, bziebart, rhc, mickunas}@uiuc.edu

Abstract. Ubiquitous computing defines a new domain in which large collections of heterogeneous devices are available to support the execution of applications. These applications become dynamic entities with multiple input and output alternatives. As a result, it is difficult to predict in advance the most appropriate application configuration. System support infrastructures for ubiquitous computing provide generic functionality to simplify the development of applications. In this paper, we present our experience in building a ubiquitous computing infrastructure. We describe lessons learnt, and explain the different problems and challenges we found during the development and deployment of the infrastructure.

1 Introduction

The evolution of Ubiquitous Computing can be compared at some level with the one of Operating Systems. Initially, before Operating Systems were developed, applications had to be handcrafted to different machines. Developers were responsible for implementing every single functional aspect, from device drivers to persistent storage access. As a result, applications were hard to port to different machines, and every application had to be built from scratch. Operating Systems provide basic services that are independent of the hardware details of every machine. Applications are not built for specific machines. They are built using the Operating Systems' system support and therefore can be easily ported. Furthermore, application developers leverage low-level functionality such as memory management, and process and thread management, which are required by all applications.

Ubiquitous computing has been following a similar path. Initially, most applications were handcrafted to specific environments and requirements. However, as the number of applications increases and researchers gain more experience in the field, a number of system support infrastructures have emerged. These infrastructures factor out common functional requirements and simplify the development of dynamic applications.

In this paper, we describe our experience in building a system infrastructure to support the development of applications for ubiquitous computing environments. We describe the different stages we went through during the development of the infrastructure, lessons learnt, and future directions. Furthermore, as a result of interacting

* These authors are supported by a grant from the National Science Foundation, NSF CCR 0086094 ITR.

with a ubiquitous computing environment on a regular basis, we identified the relevance of "application suites."

Section two describes the different stages our infrastructure went through. It provides information about the problems we found, and how we provided support for these problems. Section three describes applications suites, and how our infrastructure enables their rapid development and testing. Section three also presents an example of an application suite we built to support a seminar class.

2 Experience in Building System Support for Ubiquitous Computing Environments

Our current system support infrastructure is the result of three years of work. The goal of this – still ongoing – effort is to provide a software infrastructure to manage physically bounded ubiquitous computing environments, and to support the development of applications customized to these environments. During the three years, the infrastructure has evolved driven by a number of successful and wrong decisions, as well as by constant interaction with an experimental ubiquitous computing environment. We have fully implemented the system, which has given us valuable hands-on experience. Furthermore, it has allowed us to validate our design and assumptions. We have played three simultaneous roles. 1) Developers of the software infrastructure and applications. 2) Users of the ubiquitous computing environment. 3) Critical evaluators of the produced system and applications.

Interaction with the ubiquitous computing environment includes using devices and applications. Examples of devices are desktops, laptops, PDAs, smart watches, and touch screens; examples of applications are music and video player, slide show presenter, location, PDF viewer, ticker tape, HTML browser, speech engine, meeting and seminar attendance, user identification and authentication based on fingerprint identification devices, iris scanners, and RF badges. This constant interaction has provided the motivation to make the system more usable and has been a key contribution to the current state of the infrastructure.

The next subsections summarize the stages of our research. We start at the time we received the hardware and the ubiquitous computing lab was just a showroom for electronic devices. We conclude at the time where the lab became a useful environment with fifteen running applications and an observable sense of coordination.

2.1 Chaos and unmanageability: Device-centrism

The basic layout of the experimental lab consists of four 51-inch touch screen plasma displays, fifteen Windows based desktop computers, an audio system, two touch screens, four wireless PDAs (Bluetooth and WiFi), a video router and a video switch, a video wall, two fingerprint detectors, an iris scanner, X10 appliance and light controllers, and RF badges and detection base stations.

Before using the infrastructure, the lack of device orchestration made the environment difficult to use and maintain. Using one display and one device at a time was feasible. However, using more than one device simultaneously was difficult because user sessions were associated to the individual devices. There was no concept of a

unified environment where applications could be spread and migrated across the different devices arbitrarily. Using the lab became a tedious task. There was no logical connection among the different devices, and therefore, having a large collection of them did not make the room more useful, or in fact, useful at all. Device-centrism was a serious obstacle that hindered the usability of the environment. This state was the opposite of what ubiquitous computing is intended to be. It was far from the vision of a globally programmable environment.

2.2 Low-Level System Support

The first prototype of the system support infrastructure brought some "low-level system order" into the aforementioned chaos. It allowed us to address the ubiquitous computing environment as a single execution environment, consisting of a collection of orchestrated resources.

The system support infrastructure provides services to detect, aggregate, and manage resources that belong to the same logical computing environment. We use the boundary of the physical space to define the computing environment. Resources present in a physical space belong to the same logical computing environment. While it is possible to use other properties to define the boundary of the environment, we leave it as future work. The system infrastructure takes into account the specific properties of a ubiquitous computing environment, including hardware and software heterogeneity, mobile devices, and resources added and removed dynamically to and from the environment. We address the ubiquitous computing environment as a collection of distributed components or agents. We list the supporting services next:

- Distributed component management.
- Distributed-component presence detection based on soft-state.
- Distributed-Component database with information about distributed components present in the ubiquitous computing environment.
- Notification service to disseminate information about the state of the ubiquitous computing environment. This service is based on events and event channels that decouple the information senders from the information receivers.
- Data management support that is capable of dynamically aggregating and removing data stored in multiple devices while providing a unified file hierarchy for the ubiquitous computing environment. It uses context to customize the global data view, hiding irrelevant data automatically. Furthermore, it supports data transcoding to overcome device heterogeneity issues.
- Lightweight directory service with references to all the previous services. This service is the entry point to the functionality provided by the ubiquitous computing environment.
- Authentication service that supports different mechanisms and assigns different trust levels to entities based on the strength of the authentication method used.

The infrastructure provides facilities for system and application developers, and it effectively addresses the device-centrism problem. The functionality it provides allows managing components present in the space, regardless of their hosting device. The ubiquitous computing environment – not the individual devices – becomes the execution domain. There are several research projects that have also identified the importance of providing system support to manage and program ubiquitous computing environments [1-3].

2.3 Scripting and System Bootstrap

The system support infrastructure allowed us to develop services and applications that exploited the resources contained in the ubiquitous computing environment. However, we were still facing two problems: lack of support for component composition and complex system initialization.

The system infrastructure uses a middleware layer to ensure programming language and OS independence. We developed several C++ and Java components using the distributed component model. However, we had to use an additional C++ or Java component to encode the component composition rules. It soon became clear that this approach required too much time and effort, and therefore, we decided to add scripting capabilities to "glue" distributed components easily. We developed a library using the scripting language to simplify script developers the access to the functionality provided by the system support infrastructure. As a result, rapid prototyping became a common practice.

Initializing the system became a time-consuming and error prone task that required manually starting the basic services in one or more machines. Before starting the process, we had to ensure that previous system services were not running. Otherwise, resources in the ubiquitous environment were using different instances of the services, therefore leading to wrong behavior. Because we were using multiple computers, we had to check for running services in all of them. In our case, this required physically interacting with up to fifteen keyboards and mice, or using a switch device to share a keyboard and a mouse among all the devices (but still required manually switching to the appropriate machine). After ensuring that all previous services were not running, we had to manually start the different services. This approach was physical-space dependent, required too much low-level system knowledge, and obviously did not scale. It took us an average of twenty minutes to initialize the system, and up to one hour to initialize user services and applications we used to demo the environment. Furthermore, in case or errors we had to repeat the same process, therefore leading to hours of frustration.

Using the scripting capabilities we implemented a distributed bootstrap mechanism that automated the initialization process. We defined configuration files with information about the required system services, initial parameters, and tentative hosts. The bootstrap mechanism implements the following functionality: automatically starts the services in the specified hosts, skips instantiation of already running services (reuses them), and registers the services in the directory service. The bootstrap process reduced the initialization time to a couple of minutes. Furthermore, it made the system portable. Every ubiquitous computing environment has its own bootstrap configuration file.

2.4 Application Development and Management Support

Traditional applications are not appropriate for ubiquitous computing environments. After building a number of applications using the low-level API, we identified six patterns that were required for all applications [4]: multi-device utilization, user-centrism, run-time adaptation, mobility, context-sensitivity, and ubiquitous computing environment independence. As a result, we provided application development and management support customized to ubiquitous computing environments that automated the aforementioned patterns.

The new application support allowed us to build a total of 13 different applications in six months. Students and researchers who were not involved in the design of the supporting infrastructure built five of these applications. The new application support functionality allowed developers to concentrate on the specifics of the application functionality. All issues related to ubiquitous computing were provided by default. Besides simplifying application development, we eliminated the possibility of having multiple mechanisms for mobility, adaptation, and partitioning. This would have been the case, had every developer been responsible for providing such functionality.

As the number of applications increased, we detected the need to combine the behavior of different applications. We built a ticker tape that uses multiple displays synchronously, and found that it would be useful to use it to display information generated by other applications. For example, we were interested in displaying the name of the song currently playing, or the news headlines. We built a mechanism that leveraged the scripting language to simplify the creation of composite applications. The requirements for this mechanism were: applications must not require changes to interoperate, any two pair of applications can be combined, and application interaction rules must support a wide range of expressions [5]. The resulting mechanism met all the requirements and used scripts to encode the interaction rules. This mechanism has been used heavily and has allowed us to build sophisticated composite applications.

2.5 End-user support

Our system support infrastructure enables the construction of applications customized to ubiquitous computing environments. However, the support provided is useful for application developers, not for end users. The inter-application interaction mechanism, for example, is useful to enable application composition. However, it requires knowledge of application event formats and application component interfaces. Based on our experience, and as the number of applications built increased, it became difficult to define inter-application interaction rules. We had an increasing number of application developers, and therefore, we were forced to go over the applications' source code to learn about the events they generated and their interfaces. As a result, we defined a new supporting mechanism and a tool to enable end users to compose applications without prior knowledge of these applications. This mechanism requires application developers to provide information (stored in XML files and scripts) about their applications that the tool uses to automate the process of composing applications. End-users interact with this tool, which guides them through the process. This is part of our ongoing research.

3 Evaluation: Supporting a Seminar Scenario

As shown earlier, our system provides the necessary services to orchestrate the hundreds of heterogonous devices available in a ubiquitous computing environment, and abstract the environment with all of its contents as a single homogenous programmable entity. To allow ubiquitous computing environments to support different activities and usage scenarios, we define the idea of an "application suite." An application suite is a collection of individual applications that are coordinated in a well-defined manner to support a specific scenario (e.g., a seminar, a meeting, and a brain-

storming session). In most cases, the individual applications are loosely coupled. This allows applications to be reused for different suites to support diverse activities.

In this section we look into issues pertaining to the development of ubiquitous computing application suites. We also discuss the "seminar application suite" we built. This application suite demonstrates the flexibility of our ubiquitous computing infrastructure and serves to evaluate the system support for rapid ubiquitous computing application development.

3.1 Rapid Development Cycle

Our system provides a complete supporting infrastructure for rapid development of ubiquitous computing applications, including mechanisms for controlling the inter-application behavior. As a result, we were able to identify an application suite rapid prototype development cycle. This development cycle is illustrated in Figure 1.

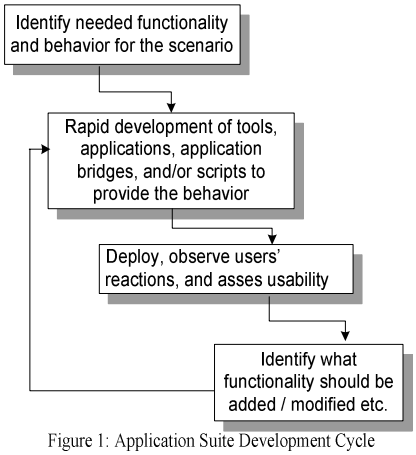

Figure 1: Application Suite Development Cycle

Ubiquitous computing environments are extremely dynamic systems with ever changing requirements and constantly refined specifications. Therefore, a prototype-based development cycle is most appropriate to cope with new demands. The first stage in our development cycle identifies the initial set of functionality and behavior needed by the scenario at hand. In stage 2, the higher-level services of our system allow the rapid development of the tools that help in achieving the functionality sought. This process may include writing new applications, or composing existing applications in special ways. In stage 3, the new tools are deployed in the test-bed environment. The usability of the tools is evaluated based on users' reactions and use. In stage 4, requirements and specifications are refined based on the evaluation step and an enhanced prototype is developed by going back to stage 2.

3.2 Seminar Format

The seminar has a total length of fifty minutes, and it is based on the discussion of two papers related to the topic of ubiquitous computing. Every week, a student chooses two papers and sends them to the seminar organizer. The organizer posts the papers on the web and sends an e-mail to all registered students. Students are then responsible for reading the papers and writing a list of paper's pros and cons. During the seminar, the student who chose the papers has five minutes to give an overall overview of the first paper. After the overview, we criticize the paper and go through a list of pros and cons provided by the students. This discussion goes for twenty minutes. The same process is repeated for the second paper. Students are only allowed to miss the class three times to get the credit. Therefore the seminar organizer must keep an attendance list for the duration of the seminar. Summarizing, the requirements for the seminar are as follows. 1. Keep attendance information. 2. A student chooses two papers every week and writes a summary for each. 3. Post the papers and notify atten-

dees. 4. Read the papers and write a list of pros and cons for each paper. 5. Discuss the papers.

3.3 Creating a Seminar Application Suite

To meet the requirements for the seminar described above, we created the "seminar application suite", which consists of five applications, and a number of application interaction rules. Table 1 shows the approximate amount of time it took us to develop the applications in the suite.

Table 1: Development Time

User identification	1 man-days
Attendance Application	1 man-days
Feedback (composite application) & scripts	1 man-days
PDF viewer	1 man-days
iCalendar	5 man-days
Total (version 1)	**9** man-days
Personal device support	1 man-days
Speech Engine & scripts	2 man-days
Grand Total	**12** man-days

The user identification application identifies users through a variety of methods (fingerprint scanners, active badges, smart watches, traditional login/password, and iris scanners). Supporting different identification methods allows the users to choose one or more convenient methods for identification. For instance, some users found biometric-based identification convenient and less obtrusive. Others however, were concerned about privacy issues and preferred to use more conventional methods, like username / password or active badges. In cases where users forget their active badge, they can resort to an alternative method. Minimal coding was needed for this service because it was based on the low-level services provided in our system, namely the notification service and the authentication service. A successful identification results in a "user entered space" notification.

The Attendance Application records the presence of people in the room under a certain context (location, time, activity). This application listens to "user entered" notifications that occur after successful identifications. This application was developed using the application support in the underlying infrastructure. By utilizing notifications, the attendance recording event is independent of the specific identification device or method used.

The ubiquitous computing environment was expected to give feedback when a user is identified. To do this, we created a composite application, where notifications from the authentication service are captured by the ticket tape application and a relevant message is displayed across the public displays in the room that included the picture of the person entering or leaving the space. The automated identification process proved to be useful in introducing new users to the rest of the class.

During the summary or discussion of the papers, users reference text or figures. The PDF Viewer is an application that utilizes our infrastructure to display multiple PDF readers simultaneously on different devices while maintaining them synchronized. All the outputs are controlled from one or more "controllers." We built the application wrapping Adobe Acrobat SDK with our application support.

As noted, the seminar involves some setup overhead, including the downloading of PDF files, firing up the PDF application with the correct files, configuring the room by running the necessary applications, scripts, and inter-application interaction, etc. A ubiquitous computing environment should provide adequate support to automate these tasks. The iCalendar application is used to schedule tasks, set context information, and configure the space according to the activity at hand. We used the iCalendar for

all the setup needed to run the seminar scenario, this included running the appropriate applications, scripts, and firing up the correct PDF files automatically at the correct time.

3.3.1. Version 2

The first version of the seminar application suite had several problems that were only apparent after testing it with new users. Displaying PDF documents on the large, wall mounted plasma displays was not very helpful, particularly, when referencing text within the document. This is because the text in general was too small and hard to read off the wall displays. We were able to address this shortcoming in version 2 by utilizing the infrastructure's support for dynamic addition and removal of personal devices to/from the space. This allowed users to bring along their laptops or tablet PCs, connect them to the space, and interact with the applications and documents running within the space. The personal devices also acted as an unobtrusive method for identifying users through stored certificates.

3.3.2. Version 3

After heavier use of the seminar application suite, it was apparent that the feedback using the Ticker Tape application was not effective, as it required users to know which display to look at for notification. It was difficult to tell if the identification process has failed (badge failed to work, or finger is misplaced on the fingerprint scanner) this is because users are not aware which display to look at for error messages etc. Further, when a large number of users enter the space at the same time, too many messages are displayed on the ticker tape in sequential order, causing an information overload on the ticket tape. In version 3 of the Seminar application suite we were able to rapidly develop a speech engine application. This application incorporates a real-time text-to-speech engine. Upon receiving selected notifications, the speech engine provides vocal feedback. By utilizing inter-application interaction and the context service, it was possible to have personalized welcome messages for users. This provided an overall better feedback experience, because it was more natural and did not require users to look at specific displays or consoles.

4 Conclusion

Our experience building a ubiquitous computing environment, and interacting with its services on a regular basis, has proven highly valuable to understand the basic requirements to support this distributed computing model. We have identified five design guidelines we consider essential to support ubiquitous computing environments: low-level system support, bootstrapping, scripting, application development and management support, and end-user support. We presented our Active Seminar application suite, which demonstrated the flexibility and generality of our infrastructure. Additionally, the Active Seminar application suites demonstrated the rapid prototyping and development capabilities that the infrastructure supports. Overall, the user experience was very positive and the space usability was intuitive.

5 References

[1] B. Johanson, A. Fox, and T. Winograd, "Experiences with Ubiquitous Computing Rooms," *IEEE Pervasive Computing Magazine*, vol. 1, pp. 67-74, 2002.

[2] P. Tandler, "Software Infrastructure for Ubiquitous Computing Environments: Supporting Synchronous Collaboration with Heterogeneous Devices," presented at Ubicomp 2001: Ubiquitous Computing, Atlanta, Georgia, 2001.

[3] C. Becker, G. Schiele, H. Gubbels, and K. Rothermel, "BASE - A Micro-broker-based Middleware for Pervasive Computing," presented at IEEE International Conference on Pervasive Computing and Communication (PerCom), Dallas-Fort Worth, Texas, USA, 2003.

[4] M. Roman and R. H. Campbell, "Providing Middleware Support for Active Space Applications," presented at ACM/IFIP/USENIX International Middleware Conference, Rio de Janeiro, Brazil, 2003.

[5] M. Roman, B. Ziebart, and R. H. Campbell, "Dynamic Application Composition: Customizing the Behavior of an Active Space," presented at PerCom2003, Dallas-Fort Worth, Texas, USA, 2003.