Aura: An Architectural Framework for User Mobility in Ubiquitous Computing Environments

João Pedro Sousa and David Garlan
School of Computer Science
Carnegie Mellon University
5000 Forbes Ave
Pittsburgh PA 15213 USA
{jpsousa,garlan}@cs.cmu.edu

Abstract: Ubiquitous computing poses a number of challenges for software architecture. One of the most important is the ability to design software systems that accommodate dynamically-changing resources. Resource variability arises naturally in a ubiquitous computing setting through user mobility (a user moves from one computing environment to another), and through the need to exploit time-varying resources in a given environment (such as wireless bandwidth). Traditional approaches to handling resource variability in applications attempt to address the problem by imposing uniformity on the environment. We argue that those approaches are inadequate, and describe an alternative architectural framework that is better matched to the needs of ubiquitous computing. A key feature of the architecture is that user tasks become first class entities. User proxies, or *Auras*, use models of user tasks to set up, monitor and adapt computing environments proactively. The architectural framework has been implemented and is currently being used as a central component of Project Aura, a campus-wide ubiquitous computing effort.

Key words: Ubiquitous computing, mobility, architectural framework, architectural style.

1. INTRODUCTION

Fuelled by Moore's Law, technology is moving towards a world populated with increasing numbers of heterogeneous computing devices, services and information sources. This emerging world of ubiquitous computing poses a number of significant challenges for software systems, and software architecture in particular.

One of the most important challenges for architectural design is to support the relatively new quality attribute of user mobility. Ideally, a ubiquitous computing infrastructure would allow users to move their computational tasks easily from one *environment*[1] to another. Moreover, users should be able to take full advantage of the local capabilities and resources within a given environment, even as other users and devices enter and leave that environment, and as resources (like available bandwidth) change [30].

Current approaches to user mobility are based on one of four techniques, none of which fully achieves these goals. One approach is to support as much of a user's computing needs as possible on a *mobile machine*. A second approach is to compute via *remote access* to a computing server that stores a users personal state and preferences, much as X-terminals do. A third approach is to provide *standard applications* that are ported to and installed in all environments. Those applications are extended to become aware of user intention and mobility. A fourth approach is to provide standard virtual platforms (such as the Java Virtual Machine) that enable *mobile code* to follow the user as needed.

There are two problems with these approaches. First, since to some degree they assume a homogenous computing baseline, they cannot take full advantage of the diverse capabilities of each environment, such as external displays, processors, and I/O devices. Second, they lack the ability to handle dynamic variations of capabilities and resources in the environment without overburdening the user with manual tuning and reconfiguration.

In this paper we propose an alternative approach that enables *mobile users* to make the most of ubiquitous computing environments, while shielding those users from managing heterogeneity and dynamic variability of capabilities and resources. Specifically, we describe an architectural framework[2] for ubiquitous computing applications with the following key features: first, user tasks become first class entities that are represented explicitly and autonomously from a specific environment. Second, user tasks are represented as coalitions of abstract services. Third, environments are equipped to self-monitor and renegotiate task support in the presence of run time variation of capabilities and resources.

As we will see, this architectural framework has a number of important benefits. By representing user tasks explicitly, we provide a placeholder to capture user intent. This knowledge is used to guide the search for suitable configurations in each new environment. By representing tasks as service

[1] In this paper, we define "environment" informally as the set of devices and applications that are accessible to a user standing at a particular location.

[2] By "architectural framework" we mean an architectural style for applications and services together with supporting run-time infrastructure (or middleware) that supports their invocation, interaction, and reconfiguration.

coalitions, the infrastructure can recognize when all the essential services in a task can be supported, instantiating them jointly, or otherwise provide early warning to the user that that is not possible. By providing an abstract characterization of the services in a task, the infrastructure can search heterogeneous environments for appropriate matches to supply those services. By providing the environment with self-monitoring capabilities, the infrastructure can detect when task requirements (such as minimum response time) are not met, and search and deploy alternative configurations to support the task.

Section 2 describes the proposed architectural framework, making it concrete how the intended features are supported by the architectural design. Section 3 illustrates the workings of the framework using a task migration scenario as an example. Section 4 details the current state of our research, discusses the benefits and limitations of what has been achieved, and outlines future research. Section 5 describes related work, while Section 6 summarizes the main results.

2. AURA'S ARCHITECTURE

The central architectural challenge in supporting computational needs of mobile users is to satisfy two competing goals. The first is to *maximize the use of available resources* – that is, effectively exploiting the increasingly pervasive computing and communication resources in our environments. The second is to *minimize user distraction and drains on user attention*.

Today, a major source of user distraction arises from the need for users to manage their computing resources in each new environment, and from the fact that the resources in a particular environment may change dynamically and frequently.

In Project Aura at Carnegie Mellon University we are developing a new solution to this problem based on the concept of personal *Aura*. The intuition behind a personal Aura is that it acts as a proxy for the *mobile user* it represents: when a user enters a new environment, his or her Aura marshals the appropriate resources to support the *user's task*. Furthermore, an Aura captures constraints that the *physical context* around the user imposes on tasks (more on this below). Examples of user tasks (or simply *tasks*) are: writing a paper, preparing a presentation or buying a house. Each of these tasks may involve several information sources and applications.

To enable the action of such a personal Aura, we need an architectural framework that clarifies which new features and interfaces are required at system- and application-level. The framework must also define placeholders for capturing the nature of the user's tasks, personal preferences, and intentions. This knowledge is key to configure and monitor the environment, thus

shielding the user from the heterogeneity of computing environments, as well as from the variability of resources.

Figure 1 shows a bird's-eye view of our architectural framework. There are four component types: first, the *Task Manager*, called *Prism*, embodies the concept of personal Aura. Second, the *Context Observer* provides information on the physical context and reports relevant events in the physical context back to Prism and the *Environment Manager*. Third, the Environment Manager embodies the gateway to the environment; and fourth, *Suppliers* provide the abstract services that tasks are composed of: text editing, video playing, etc. From a logical standpoint, an environment has one instance of each of the types: Environment Manager, Context Observer and Task Manager.[3] Although the boundaries of an environment are defined administratively, they typically correspond to some physical area, like a floor or a building. Each environment may have several service Suppliers: the more it has, the richer the environment is. Much like naming servers on networks do today, Environment Managers cooperate to find and marshal remote Suppliers when that is required by the user's task.

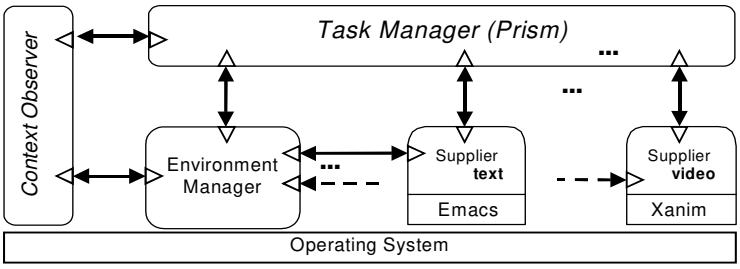

Figure 1. Aura bird's-eye view

2.1 Task Manager (Prism)

Prism embodies the concept of a personal Aura. It strives to minimize user distractions in the face of the following four kinds of change:

- The user moves to another environment: Prism coordinates the migration of all the information related to the user task to the new environment, and negotiates the task support with the new Environment Manager.
- The environment changes: Prism monitors Quality of Service information provided by the Suppliers supporting the user's task. Whenever that information becomes incompatible with the requirements of the current

[3] An environment may have redundancy of these components for the sake of robustness.

task, or the monitored Supplier fails, Prism queries the Environment Manager to find an alternative configuration to support the task.

- The task changes: Prism monitors explicit indications from the user and events announced by the Context Observer. Upon getting indication that the user intends to interrupt the current task or to switch to a new task, Prism coordinates saving the state of the interrupted task and instantiates the intended new task, as appropriate.
- The context changes: task descriptions include constraints on the context, cap