Saving Users from the Distractions of Ubiquity:
an architectural framework

João Pedro Sousa
School of Computer Science, Carnegie Mellon University
jpsousa@cs.cmu.edu

Abstract. Ubiquitous computing brings up a fundamental tension between two requirements. First, people are increasingly mobile, and so are people's activities. The pervasiveness of computing systems makes it desirable for users to take full advantage of the devices, capabilities and resources available at each different location. For that, however, users have to deal with heterogeneous devices, with different applications and with opportunistic migration of personal information. Second, users would like to focus on their real tasks, rather than wasting mental resources in dealing with the configuration of computer systems to support those tasks.

In Project Aura, we are experimenting with an architectural framework that reconciles these two requirements. The framework is based on the explicit representation of user tasks in a device- and application-independent way. This framework introduces standard components that exploit such descriptions and automatically configure the environments on behalf of the user. The framework easily accommodates legacy applications.

1 The Problem

Suppose that a user working at his office would like to continue his work in a smart room down the hall. Today, the user must checkpoint all the applications he is using, saving the manipulated files, and possibly doing some cleanup. Upon entering the smart room, the user must find and restart the required applications, making sure the manipulated files are available at the new location. Migrating the user's work gets trickier if the office and smart room run on different platforms: the user must find out which are "equivalent" applications to continue his work.

As another example, suppose that the user wants to take notes on the outline for his next talk. If resources are plentiful, the user can setup a speech recognizer in his PDA offloading the heavy recognition algorithms to a remote server. But when resources become scarcer, either because of degrading bandwidth or battery drain, the user is willing to use the PDA's text editor instead. Today, the user must monitor resources himself. Worse, he must have an understanding of how much the applications will spend in order to forecast whether or not the available resources will be sufficient to carry out his task for the expected duration.

The increasing pervasiveness of computer systems makes appealing for users to take full advantage of all the devices and resources available at each location. Mobility also exposes devices, and ultimately the user, to drastic resource variability. Therefore, a sophisticated configuration that delivers great utility to the user may be a terrible choice when resources are limited: the user may end up with a dead battery halfway through his work.

The challenge then becomes enabling users to take full advantage of the promises of Ubicomp while saving them from the distractions of managing heterogeneity, discovery of services, opportunistic access to personal information and resource variation.

2 Why Existing Approaches won't work

Remote computation: local devices act only as I/O, all computation running in a remote server. Unfortunately, good connectivity is critical for this approach to work at all – and that cannot be assured in Ubicomp. Even in circumstances where connectivity is not an issue, to deal with distinct devices, the remote services assume only the set of common capabilities to all devices. This is necessarily a restricted set of capabilities, which makes it both unsatisfactory to the user and wasteful of local capabilities.

Mobile and wearable computing: users carry as many personalized devices as they will need to perform their tasks. However, just by itself this approach doesn't address making it easier to take full advantage of other devices and resources that the user encounters in the environment.

Code migration: personalized applications and their data follow the user around. This approach makes a strong assumption about the compatibility of all the engines targeted to run the mobile code. Furthermore, mobile applications will have to deal with the diversity of devices the user will encounter. There are two possible avenues to this: either enrich the logic of the mobile applications to handle a multitude of device characteristics (with obvious software engineering problems) or assume a set of common capabilities (which leads to the same limitations as remote computation).

Standard applications: having a set of standard applications/services migrated to every device would make it easier to migrate the user's task. However, people have been trying for years to establish that in the more uniform world of desktops, but their efforts were frustrated by the roadblocks of platform diversity and investment in native applications. These roadblocks will only get worse in Ubicomp.

3 Reconciling the foes

Clearly, mobile users living in a Ubicomp age will benefit the most by using the native capabilities and local resources in each environment they roam into. However, they shouldn't have to worry about the issues that stem from migrating their work: managing heterogeneity, discovery of services, opportunistic access to personal information and resource variation.

To save the users from those distractions, the Ubicomp infrastructure must become aware of the user's location and intent, and according to those it must:

- automatically migrate user's data observing privacy policies;
- automatically marshal appropriate devices and applications to support the user's task;
- automatically configure such devices and applications;
- monitor the Quality of Service being delivered and make adjustments to the configuration as directed by adaptation policies.

The key idea to enable the infrastructure to play such role is to create an ontology of user tasks and computing services that is independent of devices and applications. For instance, if the user is writing a review of a video-clip, the task description would assert that the user needs to *play video* x and to *edit text* y. Furthermore, it would codify the status of the task, such as cursors, pane zooming, settings (e.g. spellchecking enabled), etc.

This idea is different from the existing approaches in where it assumes the common ground. Existing approaches are inadequate for Ubicomp because they assume commonality at some level ranging from device characteristics, to operating systems, to communication protocols, to virtual machines, to applications.

We assume commonality only at user level: *play video* means the same for the user, regardless of which application and device will actually be supporting it. It is up to each Ubicomp environment the user wants to resume his task on to map such task descriptions to the specific characteristics of that environment. For instance, by mapping *play video* to Media Player, by opening video x with it, by advancing the video to the specified time cursor and by enforcing the video playing settings described in the task (and that Media Player supports).

4 Architectural Framework

In Project Aura [1], we are experimenting with an architectural framework that simultaneously enables the user to take full advantage of each computing environment, while saving him from the distractions of configuring heterogeneous devices and applications [2] – see Figure 1.

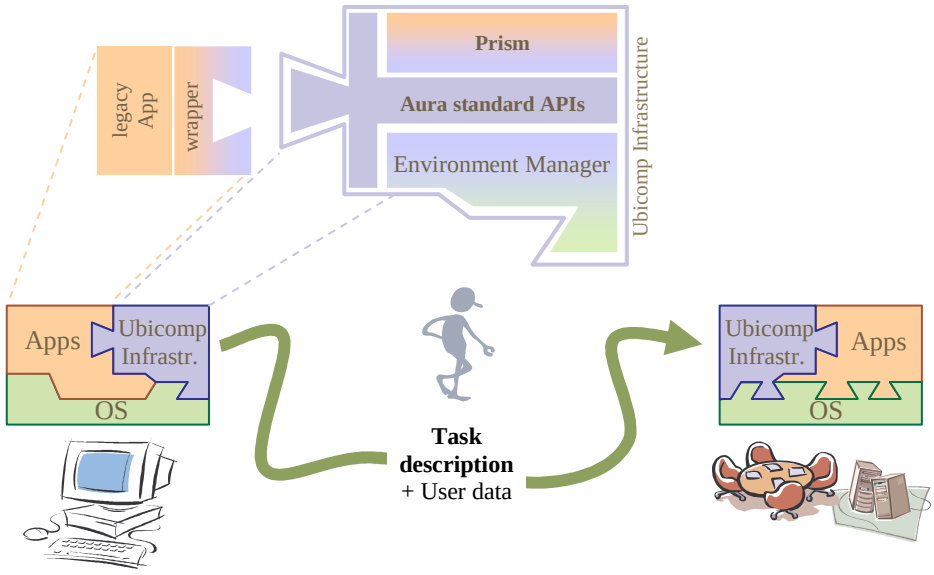

Figure 1. Migrating user tasks in the Aura Architecture Framework

The key component of this framework is Prism, who exploits platform-independent descriptions of user tasks and supervises the configuration of the computing environment to support those tasks. The role of Prism includes coordinating the automatic marshaling of devices and applications to support the user's task, monitoring QoS gauges on the marshaled configuration, and enforcing policies for resource adaptation and for privacy (e.g. by making sure confidential data is only migrated to trusted locations).

To make it easier to deploy the framework in a variety of platforms, we separated the concerns of exploiting the task descriptions, which is fairly platform-independent, and of actuating in the particular environment. The Environment Manager component offers Prism and the applications a set of standard interfaces to deal with service discovery, resource adaptation, interconnection of services and remote file access. Obviously, the implementation of these mechanisms is specific to each platform. So far we have implemented Aura's Infrastructure over Windows and Linux.

Legacy applications are easily integrated into this framework by wrapping them with Aura APIs. Such wrappers enable Prism and the Environment Manager to treat applications uniformly with respect to status checkpoints, QoS monitoring, configuration and interconnection. So far we have wrapped MSWord, Emacs, Media Player, Xanim, PowerPoint, Sphinx, Festival and Babelfish. Naturally these wrappers bridge the application's existing capabilities to Aura's standard APIs, but do not by themselves enhance the capabilities of the applications. The strategy followed to represent tasks – based on markup formats – enables user tasks to be interchangeably migrated between applications that deliver the same type of service, say *edit text*, but nevertheless offer different levels of sophistication.

By factoring out of the applications, and into the Ubicomp Infrastructure, a set of common features (like migration of user tasks, configuration awareness, resource monitoring and adaptation policies; and of course service discovery and remote file access) we hope to make it easier to develop new applications with the requirements of Ubiquitous Computing in mind.

REFERENCES

1. D. Garlan, D. Siewioriek, A. Smailagic, P. Steenkiste. Project Aura: Towards Distraction-free Pervasive Computing. *IEEE Pervasive Computing*, 1(2):22-31, April-June 2002.

2. J-P. Sousa, D. Garlan. Aura: An Architectural Framework for User Mobility in Ubiquitous Computing Environments. *Proc. Working IEEE/IFIP Conference on Software Architecture* (WICSA 2002), to appear.