Environment Mobility — Moving the Desktop Around[*]

Dag Johansen
Dept. of Computer Science
University of Tromsø
Norway
dag@cs.uit.no

Håvard Johansen
Dept. of Computer Science
University of Tromsø
Norway
haavardj@cs.uit.no

Robbert van Renesse
Dept. of Computer Science
Cornell University
USA
rvr@cs.cornell.edu

Abstract

In this position paper, we focus on issues related to middleware support for software mobility in ad hoc and pervasive systems. In particular, we are interested in moving the computational environment of a mobile user following his trajectory. We present details of WAIFARER, a prototype implementation that automatically saves and restores application level state to support this mobility. Security, integrity, and fault-tolerance are just some of the key problems that need to be addressed in the future.

1 Introduction

Pervasive computing is there and growing, but people still have to switch contexts and move their state around manually. Hence, mobile users constantly need to create and personalize their working environments once they move from one computer to another. A very common, but trivial example is a user moving current tasks from his office environment to his home. Before leaving, s/he must manually store and e-mail documents and references from the *source* computer. Similarly, upon arrival at the *destination* computer, s/he must parse the e-mails, start specific programs, open attachments, find bookmarks, maybe install some required software, integrate with local files and do manual version controlling.

We are interested in automating this type of *environment mobility*. An environment is the set of applications and services whose state needs to be captured from the source computer when a user decides to move on. For instance, if a user is editing a text document, is listening to some mp3 music in the background, and has his mailer and browser active on the desktop, the same environment should be recreated upon arrival at a destination.

In a pervasive setting, a user is moving about in a far more heterogeneous environment. He now carries some wireless connected, memory rich computer device and docks into environments along his trajectory. Extra computational, connectivity, and display resources are then borrowed or rented from the destination environment.

The rest of this position paper is structured as follows. In section 2, we present the design principles we conjecture to be important in a pervasive computing environment. Based on these principles, we are building and evaluating prototypes. Software mobility is key among these principles, and provide a state of the art survey of this area in section 3. Next, in section 4, we present some of the prototype middleware systems we have built to gain initial insights into this problem domain. In section 5, we outline the main challenges we have identified for this type of computing in a pervasive environment. Finally, in section 6, we provide a summary of this position paper.

2 Our Approach

In WAIF[1], we are investigating how to structure next-generation large-scale pervasive systems [7]. The infrastructure we build is much like in Oxygen[2] and Aura [3], but we focus more on how the next generation Internet can be made programmable and extensible with personalized, mobile software. Our goal is to replace the old, time-consuming pull-based Internet, with a push-based one delivering high-precision information in a timely manner. Servers initiating information dissemination can be programmed by accepting entrant client code and data for execution [1].

We investigate and apply four design principles in our pervasive computing environments.

[*]This project is partially funded by the Research Council of Norway (IKT-2010 Program).

[1]`http://www.waif.cs.uit.no/`
[2]`http://www.oxygen.lcs.mit.edu/`

2.1 Proactive

Principle 1 *Pervasive systems should be proactive.*

The traditional ways for users of computer systems have typically involved them to explicitly request, or pull, information when needed. As an alternative, we are building *proactive* middleware systems that *push* information to its users. Other projects like, for instance, Aura [3] also use proactivity to anticipate requests. Such a structure lends itself naturally to information overload problems, so an obvious research goal is how to achieve both recall and precision in a proactive environment.

2.2 Personal Overlay Network Systems

Principle 2 *A single user should have his private (push-based) network.*

For many years, computers systems were structured to accommodate many users per computer. In our pervasive computing model, though, we structure systems initially so that many computers serve a single user. For each user, we create a *personal* overlay network system (PONS). This ad hoc network serves a single user by filtering, fusing and pushing information based on personal preferences. In addition, a PONS provides a distributed personal file system and private compute resources.

2.3 Extensibility

Principle 3 *Mobile code should be used to achieve expressiveness and upstream evaluation.*

A publish/subscribe system like, for instance, Siena [2] achieves expressiveness by evaluating filter predicates close to data sources. We advocate a similar, but even more extreme design principle for our pervasive systems. We actually *program* the servers by deploying client code at the data sources. This resembles how we used mobile agents in TACOMA [6]. As such, a PONS can be created by extending servers with client code.

2.4 Mobile Computing

Principle 4 *Software mobility should be treated as a first order design principle.*

In a pervasive environment, users are on the move as a rule. Nevertheless, software for mobile users is still designed to be booted and run at a single computer, and never relocated. Today, you can not move a standard application in the midst of its execution.

We conjecture that applications and services should be built following the trajectory of a user, and this without necessarily moving the computer along. One of our main research goals is to devise design patterns and templates for software that can be moved. The rest of this position paper focuses on this particular problem.

3 State of the Art

Software mobility can be provided technically in a number of ways, but each approach has its limitations and specific requirements.

3.1 Move User Interface

One approach is through a remote access model, where a user logs in remotely to a source computer. Applications run at the source computer, but the desktop environment (user interface) is displayed at the destination computer.

The advantage with this approach is simplicity, but it also has its limitations. It is, for instance, not trivial just to redirect output from a running application to a new computer. Hence, applications need to be stopped and restarted from scratch from the destination computer. Since few applications are state-full, they must be manually brought back to the state they were in at the source computer. Another problem is network dependence, with inherent security, performance, and partitioning problems.

A variation of the remote computing concept is implemented by the X Window System[3] and other thin client middleware. The graphical user interface of an application can now be running on a different computer than other parts of the application. As part of logging off, all open application sessions are closed. Hence, moving to another destination implies manually restart of the same applications.

What is neat with this approach, though, is that the X Window System implements a session manager. This is used to store application state, so that a restart of the system brings the applications back to a state-full state. We have taken advantage of this concept in our work, and we refer to section 4 for how we leverage use of this protocol and session manager.

3.2 Move Hardware

A common approach is to move the source computer to the destination environment. A mobile user carries his laptop around and just plugs it into the destination infrastructure. Remote applications now need to be restarted, if possible.

[3]`www.x.org`

In the pervasive environment we are building, we assume that a pure hardware solution has its limitations. A PDA or laptop solution is a compromise for a mobile user, while we build an environment where the docking environment provides a much more powerful virtual computer. This environment is like an ad hoc network that can be leased or rented. Our portable client hardware will soon be a cellular phone equipped with WiFi capabilities and Giga-byte memory.

3.3 Move Computation Along

A third approach is to move applications about. Different types of mechanisms for application mobility have been investigated, process migration one of them. A process migration mechanism is typically an operating system service which captures the state of a running process and recreates it at the destination. Transparency is a goal so that a running process can be moved at *any* point in its execution. State capturing at the process abstraction level requires a homogeneous hardware infrastructure.

Several systems were built more than a decade ago with transparent migration support [11, 14, 16], but they never made it into real production systems[4]. There are a number of technical reasons for this, including problems with pending messages, open files, and host security. The lack of applicability for process migration mechanisms also made such techniques less interesting.

Process migration can be supported in a less transparent way. This has been demonstrated in a system like, for instance, Condor [9], where programmers manually insert application-level check-pointing and restart instructions. This gives application programmers more control when a process is ready for migration, for instance, right after a checkpoint has been taken. The check-pointed data can now be used to restart the application at another computer.

Mobile agent technologies have also been used to move applications around [4, 6, 8]. An application is implemented as one or a group of agents. The agent itself decides when to move.

Most of the mobile agent systems are implemented in Java and support agents implemented in Java. This limits the type of applications that can be moved. One exception is TACOMA, which is built for moving a group of agents implemented in almost any programming language. This also includes legacy code and binaries, which makes it useful for bringing a complete desktop environment around.

A central storage like, for instance, a distributed file system can be used to move applications around. A user stores his files upon departure from the source, and later downloads them to the destination computer. Aura uses the distributed file system Coda [10] to support such nomadic disk access. Coda has been extended with a client-close proxy that prefetches and stores volumes of data that can be accessed by the client.

[4]There is one notable exception, MOSIX (www.mosix.org), but this system is specialized for parallel computation environments, not for personal computing.

3.4 Move Data Along

A common approach for moving a desktop environment around is to move meta-data and application data, but not the applications. This assumes that applications are already installed at the destination. This is what many Internet users do frequently, but manually, by zipping data and e-mailing files around, or sometimes using the check-in/check-out support of a version control system. The user either knows which applications to start at the destination (i.e. his home computer), or the applications are started automatically by opening attachments of specific types.

This concept has been automated in Aura by introducing the *task* concept. A task is an abstraction layer above applications, but below the user. Its role is to explicitly represent user intent so that Aura can adapt to or anticipate user needs. User tasks are explicitly represented as coalitions of abstract services, so that application data can be check-pointed through Coda for later restart at another computer. This way, pervasive applications like, for instance a standard editor or video-player, do not have to be moved, but are activated at the destination with application data as input.

4 Our Approach — WAIFARER

A WAIFARER client moves among environments equipped with extensible servers. Docking into an environment typically involves off-loading client computations and running them on an ad hoc network of extensible servers.

One example of an off-loaded computation can be proxy software connecting back to a source environment. Using the destination environment for connectivity might save both battery capacity in the client computer and give better network bandwidth.

Another example is to use CPU-cycles in the destination environment. The client can off-load and build, for instance, an ad hoc, personal grid.

4.1 Data Mobility — Desktop Migration

We have built a series of WAIFARER clients supporting software mobility. A first prototype uses *data mobility* and moves tasks around. This gives the user the same desktop environment whenever he touches base with a new environment.

Upon departure from the source computer, state from applications like, for instance, mp3 players, games, and text editors are automatically hoarded and marshaled to an XML-file.

Next, a USB-memory stick is used for transport between the environments [13], but the marshaled state can also be transferred using, for instance, e-mail or FTP. When the USB-memory is plugged in at the destination computer, the same applications are restarted with the check-pointed state. An mp3 song, for instance, is restarted with a specific offset that was set by the checkpoint mechanism.

Our scheme assumes that the applications already exist at the destination, a valid assumption today. On most computers, you find the same set of standard browsers, editors, e-mailers, mp3 players, games and the like. Another limitation with our prototype is that we needed to modify applications with checkpoint-restart abstractions. Because of these limitations, our implementation of this approach currently only supports Python applications linked with a library of our checkpoint-restart abstractions.

4.2 Wrappers — Desktop Migration

A second prototype implementation also provides task migration, but applications do not have to be instrumented with our checkpoint-restart abstractions.

If an application does not write necessary state to a file, it can be captured and recreated by wrapping techniques. This means that a wrapper binds to an interface exported by the application, and uses this to capture and restart it. Such API's are commonly found in component-based applications, such as those using Microsoft COM, or Gnome's Bonobo component models. Still, for this approach to work, it is required that the application exports functionality which allows the wrapper to extract and set the correct type of information.

We have created wrappers for the most common Microsoft applications [12]. Associated with applications like, for instance, Powerpoint, Internet Explorer, or Microsoft Word, is a COM object. Through COM objects, we manage to capture enough state for later recovery.

The wrappers store state from the applications in a *task description file*. This includes a description of the task (i.e. volume of a music file and its offset), requirements (i.e. format that the music player must support), and application data (i.e. an mp3 song). Upon marshaling, the task description file is stored in a distributed file system (or spooled in a mail system like, for instance, smtp or pop) for later recovery. This Python based file system implements file operations as a web service for easy access from any remote platform or programming language. Enabling the web service with SSL for security is easily done, but it has impact on performance and requires that all clients support SSL.

The main disadvantage with our wrapper approach is that each application requires a specific wrapper. Also, generating a task description is not trivial, especially since no standard for this exist.

4.3 Move Legacy Code Transparently

Our initial WAIFARER implementation and the one.world Java framework [5] taught us the same lesson, that applications written for mobility can easily be migrated. However, such approaches exclude the large corpus of existing applications not built for mobility. In addition, this approach is to restrictive with regard to what programming languages, libraries, and platforms an application programmer might choose from.

Therefore, we stress even more the potential ability to move legacy application in our current WAIFARER implementation[5]. Our goal is that most existing applications should be movable without any, or minimal changes. Even if no API for moving an application around is supported by these legacy applications, we are investigating existing API's and protocols to see if we can apply them for our purpose.

In particular, the “X Session Management Protocol” (XSMP) [15] is an interesting candidate for our migration problem. This protocol is a well established X-Consortium standard and is employed by several popular Linux and Unix desktop environments, like KDE and Gnome.

XSMP introduces the concept of a persistent session of running applications. That is, if an application needs to be terminated as a consequence of a user logout, a session manager (SM) will ask it to save its state and terminate. The application is also required to provide the SM with a restart command, which, when executed, will bring the application back to its former state.

A SM is required to provide a private data storage to each application. While the XSMP does not impose any limitation on the amount or type of data that a client can store in the SM, transferring large state, like a mpeg video, can be impractical. A more common usage is for an application to write its state in a local file and then store only the path to this file in the SM. Most applications store this path as part of the restart command. For instance:

```
RestartCommand =
  'editor --id=ar93 --state=/tmp/ar93'
```

The XSMP SM does not have to reside on the same host as the application it manages. However, XSMP does not explicitly support migration. By using XSMP, the current WAIFARER prototype supports migration of many XSMP

[5]An initial implementation of WAIFARER is in the public domain (`http://www.sf.net/projects/waifarer/`).

enabled application by relocating their state files. This includes applications like, for instance, gedit, kedit, gthumb and nautilus.

In addition, we are also exploring how to take advantage of recovery mechanisms in, for instance, Microsoft software for migration purposes. Upon failure in, for instance, Microsoft Word, a recovery file has already been created transparently. This file can potentially be restarted on another node.

5 Challenges and Open Problems

Solving the software mobility problem for a few applications, as we have done, is a good way to get an understanding of the general issues and problems involved. Next, we need to go beyond those few examples and devise a general-purpose toolbox that (almost) automatically converts applications for being able to move from node to node. Currently, programmers who wish to use application-level check-pointing must analyze and instrument their code manually. A run-time system should provide this transparently.

The middleware system for this type of mobile computing needs to support more than just state checkpoint-restart operations. For instance, we need a way to do version control. We already experience the complications of having our environments instantiated in multiple places and us forgetting to move our memory sticks around. Also, there may be multiple ways of communicating the state (memory stick, fast internet connection, e-mail, web pages, distributed file system) that need to be explored and compared.

So far, we support only stand-alone applications for a single user. Collaborative applications, or applications that have a persistent network connection to some service, adds complexity. How to move such applications around must be investigated. Also, there are applications that should not be mobile, and we need to identify these. Due to, for instance, security restrictions, licensing agreements or data locality, some applications are better off by not being mobile.

Non-functional aspects like, for instance fault-tolerance, trust, security, performance and implications of heterogeneity are also open problems. This includes more than, for instance, traditional network security problems, with host and mobile code integrity problems as examples. Heterogeneity is also more complex, with a simple example being a text document being edited on different platforms (home-computer $\rightarrow$ PDA $\rightarrow$ office computer) or some music in one format played on a RealPlayer being restarted on a Microsoft Media Player in another format. The offset of the file is then used, but the music sampled in the supported format might need to be located and downloaded.

Context awareness is the concept of sensing and reacting to dynamic environments and activities. Important challenges are related to interaction with the user. Capturing user presence in an environment, especially combined with transparent capturing of user intent, is an example of a complex human-computer problem. Resource discovery and management must also be investigated. Since we have moved the mobility abstraction to the task level, we need mechanisms for locating and even installing applications and services on the fly.

6 Summary

We are building extensible middleware technologies for large-scale pervasive environments. Our thesis is that a user should be connected to his personal overlay network system, a PONS. Part of the PONS should have potential for relocation while in execution. This way, PONS software can be dynamically moved along the trajectory of a mobile user.

Our current run-time system requires only minimal input from the programmer. We are currently exploring how APIs and protocols of legacy applications can be utilized for this. Our three first WAIFARER implementations taught us that applications should be made with mobility in mind. That is, mobility should be a first order design principle for any pervasive application.

References

[1] I. Arntzen and D. Johansen. A programmable structure for pervasive computing. In *Proceedings of the IEEE/ACS International Conference on Pervasive Services (ICPS'2004)*, Beirut, Lebanon, July 2004. To appear.

[2] A. Carzaniga, D. Rosenblum, and A. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19:332–383, August 2001.

[3] D. Garlan, D. Siewiorek, A. Smailagic, and P. Steenkiste. Project Aura: Toward distraction-free pervasive computing. *IEEE Pervasive Computing*, 1(2):22–31, April 2002.

[4] R. Gray. Agent Tcl: A transportable agent system. In *Proceedings of CIKM Workshop on Intelligent Information Agents*, Baltimore, MA, USA, December 1995.

[5] R. Grimm, J. Davis, B. Hendrickson, E. Lemar, A. MacBeth, S. Swanson, T. Anderson, B. Bershad, G. Borriello, S. Gribble, and D. Wetherall. Systems directions for pervasive computing. In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, pages 147–151, Elmau, Germany, May 2001.

[6] D. Johansen, R. van Renesse, and F. Schneider. Operating system support for mobile agents. In *Proceedings of the 5th IEEE HOTOS*, Orcas Island, Wa, USA, May 1995.

[7] D. Johansen, R. van Renesse, and F. Schneider. WAIF: Web of asynchronous information filters. In *Lecture Notes in Computer Science: "Future Directions in Distributed Computing"*, volume 2584. Springer-Verlag, Heidelberg, April 2003.

[8] D. Lange and O. Mitsuru. *Programming and Deploying Java Mobile Agents with Aglets*. Addison Wesley, 1st edition, 1998.

[9] M. Litzkow, M. Livny, and M. Mutka. Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, pages 104–111, San Jose, CA, USA, June 1988.

[10] L. Mummert, M. Ebling, and M. Satyanarayanan. Exploiting weak connectivity for mobile file access. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, pages 143–155, Copper Mountain Resort, CO, USA, December 1995.

[11] M. Powell and B. Miller. Process migration in DEMOS/MP. In *Proceedings of the 9th ACM Symposium on Operating Systems Principles*, pages 110–119, Bretton Woods, NH, USA, October 1983.

[12] A. Søreng, A. Johannessen, and K. Pedersen. Task migration. INF-3203 project report, University of Tromsø, Tromsø, Norway, April 2004.

[13] J. Tennøe. WAIF — task migration. Master's thesis, University of Tromsø, Tromsø, Norway, December 2003.

[14] M. Theimer, K. Lantz, and D. Cheriton. Preemptable remote execution facilities for the V-system. In *Proceedings of the 10th ACM Symposium on Operating Systems Principles*, pages 2–12, Orcas Island, WA, USA, December 1985.

[15] M. Wexler. *X Session Management Protocol. X Consortium Standard. X Version 11, Release 6.4*. Kubota Pacific Computer, Inc., 1994.

[16] E. Zayas. Attacking the process migration bottleneck. In *Proceedings of the 11th ACM Symposium on Operating Systems Principles*, pages 13–24, Austin, TX, USA, November 1987.