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 or her 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.

Categories and Subject Descriptors

H.1.m [**Models and Principles**]: Miscellaneous

General Terms

Design

Keywords

Pervasive computing, mobile code

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 a mailer and a 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. S/he now carries some wireless connected, memory rich computer device and docks into environments along the trajectory. This docking is typically done using wireless networks (IEEE 802.11). Extra computational, connectivity, and display resources can then be borrowed or rented from the destination environment. This implies that computations can be offloaded from the portable device. Alternatively, computations can be pulled down over the network from a remote source computer for execution. Consequently, this environment acts as a virtual computer for the docked client.

The rest of this position paper is structured as follows. In section 2, we present our pervasive computing approach. Focus is on programmable servers, where software mobility is used as a fundamental structuring technique. We 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 [6]. Our infrastructure is similar to that of Oxygen[2] and Aura [2], but our focus is on how the next generation Internet can be made programmable and extensible with personalized, mobile software.

For each user, we create a (distributed) virtual computer by configuring a *personal* overlay network system (PONS). This ad hoc network serves a single user by filtering, fusing, and pushing information based on personal preferences (see Figure 1). In addition, the PONS provides a distributed personal file system and private compute resources.

A publish/subscribe system like, for instance, Siena [1] 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 [5]. As such, a PONS can be created by extending servers with client code.

*This project is partially funded by the Research Council of Norway (IKT-2010 Program, No. 152956/431).

2nd Workshop on Middleware for Pervasive and Ad-Hoc Computing Toronto, Canada

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

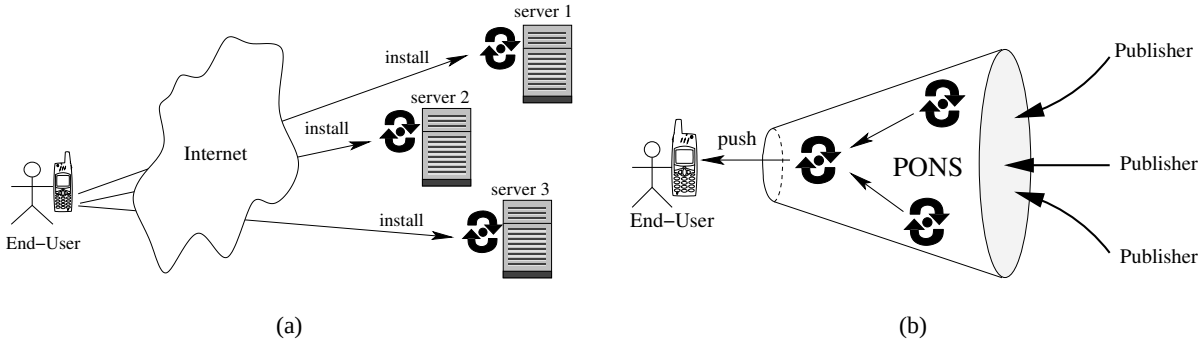

Figure 1: By installing code at data sources (a), the end-user constructs a personal overlay network system (PONS) that pushes information based on personal preferences (b).

Such a structure lends itself naturally to solving information overload problems. By allowing servers that disseminate information to be programmed, we expect both recall and precision requirements of end users to be met.

In a pervasive environment, users are on the move as a rule. Also, different computers serve the same user at different times. 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.

There are many options for doing so. At one extreme, the applications can be designed to support saving and restoring state. For example, an MP3 player's state could consist of the song being played, the offset into the song, the volume, etc. Alternatively, the operating system could support migrating processes, transparent to those processes themselves. This would involve moving all of the state of the application including the code, the run-time memory, and the PC registers.

One can even imagine transfering the entire operating system state, and virtual machine monitors available today make this possible [7]. This would involve a lot of state, but small devices like iPods have sufficient storage and I/O capacity to do so. Alternatively, if there is a fast network available between the source and destination host, it may not be necessary to move any state and simply provide a thin client interface to the source computer.

We conjecture that applications and services should be built following the trajectory of a user, and this without necessarily moving the computer along. Which is the best approach for doing so depends on many factors, and one of our main research goals is to devise design patterns and templates for software that can be moved.

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.

A variation of the remote computing concept is implemented by the X Window System[3] and other thin client middleware such as VNC [4]. The graphical user interface of an application can now be shown on a different computer then the running application. The fact that the user is located on a remote computer is masked out by the middleware.

What is neat about this approach is that it allows applications to run uninterrupted and unaware of user movements. This property is important for applications that can not easily be restarted. It also does not require any explicit support in the application. Another advantage is that there is no need to relocate application state.

Many applications utilize hardware other than a screen and keyboard. For instance, an MP3 player uses a soundcard, and the sound must now be redirected transparently to a loudspeaker close to the user. Similar problems exist for other hardware like, for instance, printers and cd-writers. Providing full end-to-end transparency for all hardware increases the complexity of the middleware system dramatically.

In addition to the complexity incurred by providing a higher level of end-to-end transparency, the remote access approach also has issues related to network bandwidth and latency. Since the local data-bus (e.g. PCI) is faster than current commodity networks, application are generally not designed to be economical about data-bus bandwidth. This is problematic when the local data-bus traffic needs to be routed through a much slower network connection. To cope, the end-user experience is traded for lower bandwidth by employing techniques like colour reduction, size reduction, lossy compression, etc. In some cases, like with a dvd-player, the bandwidth constraint of the network can often not be meet without a dramatically reduced quality output. Also, the higher latency of a network connection can render interactive intensive applications like, for instance, a paint-program unusable. For these reasons, a thin-client, remote access model does not perform well over a wide-area network.

3.2 Move Hardware

A common approach for environment mobility, 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.

This approach is neat as it provides the user with the same computing environment at all destinations. That is, the same hardware and software platform will be available wherever s/he goes. Also, a large amount of application state can be brought along.

Users often need to make use of resources in the destination in-

[3] http://www.x.org

[4] http://www.realvnc.com/

frastructure. Underlying middleware can discover and make such resources available to the applications. In some cases, however, this might require applications to be restarted.

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, so 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.

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[5]. 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 checkpointing 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 [3, 5, 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 down-loads 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.

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.

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

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 s/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.

This 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 this 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 does not have to be instrumented with our checkpoint-restart abstractions. This is advantageous for applicability purposes since legacy applications now can be moved about.

We use wrapping techniques to achieve this. 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 the Gnome Bonobo component model. 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 an application like, for instance, Powerpoint, Internet Explorer, or Microsoft Word, is a COM object. Through COM objects, we manage to capture enough state

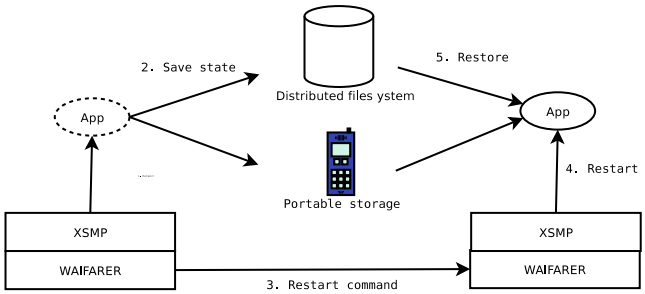

Figure 2: Migrating an application.

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 [4] taught us the same lesson: that applications must be designed for mobility. However, these approaches exclude the large corpus of existing applications not built for mobility. In addition, they are too 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 prototype[6]. 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. As such, XSMP enabled applications, like KEdit, GThumb, GEdit and Nautilus, already have a checkpoint–restart mechanism embedded.

A SM is required to provide 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, for instance, 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. The following illustrates this option:

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

While the XSMP SM does not have to reside on the same host as the application it manages, the protocol does not explicitly support migration. However, by utilizing the XSMP checkpoint-restart mechanism, our WAIFARER prototype is able to migrate applications.

Upon receiving a migration request from the end user, WAIFARER uses the XSMP protocol to checkpoint and stop the application (Figure 2). The application state is made available to the remote computer, either by a distributed file system or a portable storage. The source WAIFARER then sends a message to the remote WAIFARER containing the restart command provided by the application. The remote computer can then restore the application by executing the restart command.

WAIFARER maintaisn a global view of computers and running applications available to an individual user. This enables the user to both pull remote running applications to his or her current location and to push local running applications to a remote host.

The ability to both pull and push applications is important. If, for instance, the user needs to relocate to a known destination computer, the application can be pushed to the new location while the user is in transit. This will reduce the boot time at the destination. However, if the user accesses a computer where no user state has been previously pushed, the pull mechanism allows the user to retrieve applications left behind.

While the initiation of application migration is a manual process in our current prototype, it can be automated by predicting user movement through sensors in the user's environment. Also, boot time can be reduced by check-pointing and pre-copying application state to remote computers the user is likely to visit.

In addition to the XSMP protocol, 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 per-

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

sistent 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 and hardware 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 → PDA → 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 or her personal overlay network system, a PONS. This is a private virtual computer implemented by a group of extensible servers. A user can install personalized code at these remote servers, and we are in particular interested in personalized data filtering for high-precision alerts.

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] 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.

[2] 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.

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

[4] 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 IEEE Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, Elmau, Germany, May 2001.

[5] D. Johansen, R. van Renesse, and F. Schneider. Operating system support for mobile agents. In *Proceedings of the 5th IEEE Workshop on Hot Topics in Operating Systems (HotOS-V)*, Orcas Island, Wa, USA, May 1995.

[6] 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.

[7] M. Kozuch, M. Satyanarayanan, T. Bressoud, C. Helfrich, and S. Sinnamohideen. Seamless mobile computing on fixed infrastructure. *IEEE Computer*, 37(7):65–72, July 2004.

[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 IEEE 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.