An Architecture to Support Adaptive Mobile Applications

A Thesis Submitted to the College of

Graduate Studies and Research

in Partial Fulfillment of the Requirements

For the Degree of Master of Science

in the Department of Computer Science

University of Saskatchewan

Saskatoon, Saskatchewan

by

Xiaozhen Cao

Spring 2003

Permission To Use

In presenting this thesis in partial fulfillment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science
University of Saskatchewan
Saskatoon, Saskatchewan, Canada
S7N 5A9

Abstract

The increasing number of mobile users and innovative new mobile devices such as Web-enabled cellular phones or pocket PCs are placing new requirements on mobile applications. Mobile applications in the new era must have the ability to provide a unique service to users who want to access data from anywhere at any time using any device.

To handle heterogeneous mobile devices operating on a network, a new server-based architecture is proposed to support adaptive mobile applications. In this architecture, middleware technology is used to provide an intermediary between adaptive mobile applications and the network, with the primary goal being to provide adaptation services. A prototype has been built using Java^{TM} to support a range of mobile applications, including a Web server, a video server, and an FTP (File Transfer Protocol) server, with three different client devices: a PIII desktop computer, a laptop computer both with a 1024x768 pixel display, and a pocket PC with a 240x320 pixel display. The server application retrieves resource information from the middleware, applies adaptation strategies and sends adaptation data to heterogeneous client devices over different network conditions ranging from 100 Mbps wired Ethernet (IEEE 802.11b) LAN (Local Area Network) to 11 Mbps wireless Ethernet LAN to 20 kbps CDPD (Cellular Digital Packet Data) WAN (Wide Area Network) service. The prototype shows that this architecture works well to support various mobile devices including a pocket PC, a Compaq iPaq^{TM}. The analysis also shows that this architecture is a simple, general, extensible and cost effective solution to support adaptive mobile applications.

Acknowledgements

I would like to acknowledge Dr. Rick Bunt for his guidance, encouragement and assistance throughout this research project. His knowledge, his tremendous research experience, and his enthusiasm for the work were of great benefit to me. In addition, I wish to thank Dr. Carl Gutwin, Dr. Ralph Deters and Dr. Glen Watson for their advice.

I also wish to thank the members of our networking group, especially Dr. Dwight Makaroff, who spent a large amount of extra time advising me and reading my material. Wayne Wang, Tianbo Kuang and David Schwab helped clarify my design ideas. Alan Fedoruk previewed all my first drafts and gave me many useful suggestions.

My thanks also go to the Natural Sciences and Engineering Research Council of Canada (NSERC) and Telecommunications Research Laboratories (TR*Labs*), for providing financial assistance for this research.

The technical support from Greg Oster and Cary Bernath of the Department of Computer Science, U of S, and Jack Jonson and Trevor Hamm of TR*Labs* is also greatly appreciated.

Finally, I'd like to recognize my family. My parents, Jingbo and Ruiyu, who bestowed on each of their children a love for learning, my sister, Sylvia, without her help, I wouldn't have had the chance to study in Canada, and my lovely niece, Pamela, who helped prepare my presentations.

To my parents, Jingbo Cao and Ruiyu Zhao.

Table of Contents

List of Tables

List of Figures

List of Acronyms

AAC	Advanced Audio Coding
ATM	Asynchronous Transfer Mode
CDMA	Code Division Multiple Access
CDPD	Cellular Digital Packet Data
DPCM	Differential Pulse Code Modulation
EDGE	Enhanced Data for GSM Evolution
FDDI	Fibre Distributed Data Interface
FTP	File Transfer Protocol
GPRS	General Packet Radio System
GPS	Global Positioning Services
GSM	Global System for Mobile Communication
HDTV	High-Definition Television
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IP	Internet Protocol
IPAD	Inter-Personal Awareness Device
ITU	International Telecommunications Union
JDK	Java Development Kit
JMF	Java Multimedia Framework
JPEG	Joint Photographic Experts Group
JRE	Java Runtime Environment
JVM	Java Virtual Machine
kbps	kilobits per second (1 kilobit = 1000 bits)

KB	KiloBytes (1 Kilobyte = 1024 bytes)
Mbps	Megabits per second (1 Megabit = 10^6 bits)
MB	MegaBytes (1 Megabyte = 2^{20} bytes)
Microsoft SDK	Microsoft Software Development Kits
MP3	MPEG, audio layer 3
MPEG-4	Moving Picture Experts Group version 4
LAN	Local Area Network
PAN	Personal Area Network
QoS	Quality of Service
RTCP	Real-Time Control Protocol
RTP	Real-Time Protocol
RTSP	Real-Time Streaming Protocol
RTT	Round Trip Time
TCP	Transmission Control Protocol
WAN	Wide Area Network
WAP	Wireless Application Protocol
WCDMA	Wideband CDMA
WML	Wireless Markup Language

Chapter 1

Introduction

The need for communication technology to support geographic mobility of users began in the 90's. This was followed by similar requirements for computing technology. The demand for *mobile computing* is increasing dramatically, fuelled by the rapid development of technology in mobile devices and information access and the increasing availability of applications and services for mobile users. This new paradigm enables users to access the information and services they require from anywhere at any time. Inexpensive devices make mobile computing both feasible and practical. To achieve good end-to-end service across a wide range of hardware configurations or communication variations requires the ability to adapt to changing circumstances. Since the applications are the end points of a connection, adaptation solutions are most appropriately applied in the application layer [4].

The increasing demands for mobile computing have stimulated the development of the technology, but challenges remain. The main paradigms include improving network connectivity (e.g., high speed wireless technologies), modifications to network protocols (e.g., mobile IP (Internet Protocol) [41]), and improving capabilities and portability of mobile devices.

The two most significant improvements in the past decade are related to the computational ability of mobile devices and wireless technologies. The ability of wireless interfaces to access the Internet, as one example, makes small mobile devices far more powerful and versatile than mere organizers. The notebook computer, the handheld PC, the pocket PC, and the Web-enabled cellular phone will be introduced in detail in Chapter 2. Increased wireless communication bandwidth can support not only traditional applications, such as paging, messaging and email, but also a new group of mobile applications such as video telephony, on-line conferencing, on-line education systems, on-site detection systems, etc. This group of emerging mobile applications can take advantage not only of a network connection, but also of multimedia data.

These improved technologies lead to a significant complication: heterogeneity. Various mobile devices have different capabilities, different requirements and connect to the Internet through a variety of communication methods. The current capabilities of some of these mobile devices vary in quite a few aspects, from memory size to processing speed to battery life. There are also huge variations in data transfer rates, ranging from less than 28.8 kbps to more than 100 Mbps. Popular mobile devices include the BlackBerry™ produced by RIM and the Palm™ handheld devices produced by 3Com. This research focuses on one type of pocket PC – the iPaq™, produced by Compaq. All these mobile devices have different capabilities. These variations will be reviewed in Chapter 2.

A goal of mobile computing research is to make any changes that might occur in location or type of network connection transparent to the user (or at least to the mobile applications), thus allowing the mobile users to work in the same manner and with the same productivity no matter where they are, how they are connected, or which platform they are currently using. But present support for adaptation is limited. Part of the difficulty of adaptation in the mobile environment is the requirement not just to deliver data and services over challenging network conditions, but to deliver them in formats suitable for the devices that need them [4].

With the introduction of larger, more variable, more complex, and rapidly growing networks that must meet ever increasing demands, trying to alter the existing networks and protocols is not a feasible solution. Even if it were possible to develop a new adaptive network to replace today's network infrastructure, (i.e., the Internet), despite the challenges in the distribution, decentralization and large quantity, problems would still remain. The goal of adaptive networking is to provide good end-to-end service, where the end points are located in applications. Without considering the needs of applications and their users, no adaptive solution at the network level alone can solve the entire problem [4]. Any new application with new adaptation requirements might fail in such a network, and so finding adaptation solutions in application layers is crucial.

To build an effective and efficient adaptive application requires awareness of both client capabilities and changes to network conditions, and the ability to adapt the behaviour of the application as circumstances change. But the present support for adaptation is limited. Research has been done on applying the adaptive capability to a specific class of applications [1, 12, 13], and on providing operating system support without modifying existing individual applications [27, 15]. The former approach, referred to as *ad hoc* adaptation, requires no support from the operating system as all of the adaptation is the responsibility of the individual application. It lacks central control over potentially incompatible resource demands of different applications to enforce limits on resource usage. It also makes applications more difficult to implement and to upgrade. The latter approach, referred to as *application-transparent* adaptation [39], puts all the adaptation responsibility on the operating system. The drawback here is that there may be situations in which the adaptation performed by the operating system is inadequate or even counter-productive.

To overcome the drawbacks of those two approaches, researchers have tried to find a balance between them. An example is Odyssey [38], a set of operating system extensions to support mobile applications, which is designed to provide adaptive control from both the operating system and the applications. But these studies considered dynamic resources only, such as network conditions. The previous work has make a good start in improving the adaptivity to the network conditions, but lacks consideration of variation of the capability of the mobile devices such as display size and display type. The increasing heterogeneity of the mobile devices means taking these variables into account is important as well.

There is also an important design issue on where the adaptation logic should sit. Adaptation logic can be located either at the server, at the client or at the proxy. These are called server-based adaptation, client-based adaptation, or proxy-based adaptation, respectively. For reasons to be discussed later, including reduced implementation complexity and ability to deal with heterogeneous mobile devices, server-based adaptation is the focus of this research.

The previous research is reviewed in Chapter 3. The main design decisions are discussed in Chapter 4. A new server-based architecture is proposed in Chapter 5 to support heterogeneous end devices under different resource conditions. Middleware is designed to serve as an intermediary between the adaptive mobile applications and the network, with the primary goal being to provide adaptation services. The resource information that is required by diverse adaptive mobile applications is divided into static information and dynamic information. The objective is that the resource information would constitute a service provided by the operating system infrastructure instead of manipulated by each application itself. In Chapter 6, a prototype of this model, built in Java, is used to study how to monitor resources, how to organize/use the resource information, and how to take advantage of various application specific adaptation strategies. Chapter 7 shows the testing on the prototype implemented and Chapter 8 evaluates the architecture by analysis. Chapter 9 shows the possible extensions of the prototype and of the architecture design. Finally, Chapter 10 draws some conclusions from this research and proposes some areas for further study.

Chapter 2

Background

This chapter describes the context of this research. The mobile computing environment considered for this research will be described. Current technologies and applications to support mobile computing will also be introduced.

2.1 Mobile Computing

There have been various definitions for *mobile computing*, ranging from "computing while moving" [17, 28] to "location-independent computing" [24] to "ubiquitous computing" [52] [22, 21]. A brief story will help to describe the mobile computing environment envisioned in this research. The following is a hypothetical scenario of a business trip in the near future, modified from Satyanarayanan [43]:

> Ms. Mobile is sitting at her office desk, editing a report stored in a shared file system using a small notebook computer with a larger and more comfortable display and keyboard via a desktop infrared link. This task is under a group project environment for a group of collaborating and interacted users. The user-awareness and task-awareness application she is currently using provides security and integrity guarantees for shared objects and communication, and a workspace, in which users are aware of one another while working on shared objects. Soon it is time to leave for the airport.

On the way to the airport, Ms. Mobile continues her work. Her notebook recognizes that it is no longer on a LAN (Local Area Network), but continues communicating with the server via a cellular modem. The application still treats her as one of the group members and provides her with whatever she needs in the application. She finishes her editing, saves the file and announces to her group members that they can review her work. She then begins working on the slides for her seminar in Paris.

Upon arrival at the airport, Ms. Mobile boards her transatlantic flight and continues working. Although each seat is equipped with an outlet for air-to-ground telephone service, her notebook inquires and discovers that telephone charges are very high. It therefore wisely decides to let her operate disconnected and to defer all communication until she has landed.

When Ms. Mobile arrives in her Paris hotel room, her notebook discovers that the hotel's late-night telephone charges are low, and that there is a HDTV (High-Definition Television) set in her room. It therefore propagates the changes she has made so far, fetches new versions of some of the files she has cached, picks up her mail, and uses the HDTV set as her display. She works late into the night, putting the finishing touches to her slides.

The next morning, she presents her seminar. Her notebook senses the presence of a large wall-sized display in the conference room, and shows her slides on it. Her talk is about a new piece of user-interface software, and she is able to give a live demo of it using the notebook.

Once her business is completed, she decides to play tourist for a day before returning home. The concierge at her hotel provides her a hand-held context-aware tourist guide. The information presented to her is tailored based on her selections. A user profile and contextual information, including the unit's physical location, is created in the tourist guide system. The system monitors Ms. Mobile's current position while she walks around and sends relative information of the attractions nearby. She indicates items of interest on the guide. A short video describing the unique historical and architectural features of the site is seen, and the accompanying audio commentary is heard. As she passes through a major shopping district, advertisements of sales (translated into Chinese) pop up on her display. One of these interests her, and she walks into the store and purchases a gift.

Coming out of the store, she finds that she is lost. The system determines directions after she indicates some nearby landmarks from the screen.

She continues her walking tour for many more hours. Exhausted, she decides to take the metro back to her hotel. On the metro, she watches CNN news on her notebook. From time to time, as the train goes through regions of poor reception, the displayed image decreases its fidelity, i.e., from full-motion colour to slow-scan black-and-white.

The next morning, she heads for the airport and boards the flight home.

Figure 2.1 shows the abstract model of the mobile computing environment under consideration. In this model, there are diverse client devices connected to the server via the network. It is possible that one or more hops exist in the network or one or more subnetworks exist in between. The thickness of the lines denotes the bandwidths in use. A dashed line represents a wireless connection while a solid line represents a wired connection.

Mobile computing as defined in this research is different from paradigms embraced by terms such as *wireless computing*, *ubiquitous computing*, and *distributed computing*, although there is some overlap among these paradigms.

Wireless Computing There are significant difference between wireless computing and mobile computing. Mobile computing is nomadic (or location-independent) computing, supporting users who work from multiple locations, and whose means of “connection” to the server may take different forms, maybe wired or wireless, at different times [28]. Wireless computing emphasizes the wireless techniques that connect the hosts in the network, is concerned more with lower layers of network support than with higher layers, and is focussed more on hardware technology than on software. Wireless computing is not necessarily mobile in the case of building a network in some urban areas using wireless links, for example. On the other hand, mobile computing does not necessarily use wireless connections. But wireless computing is clearly relevant to mobile computing since the only practical way to connect to a network while moving is using a wireless link. Improvements in wireless computing techniques will dramatically increase the mobility and impact the trends of mobile application infrastructures and designs. At the same time, the needs of mobile users will significantly stimulate the development of wireless computing.

Ubiquitous Computing The term “Ubiquitous Computing” (Ubicomp) was first articulated by Mark Weiser in 1988 [51] at the Computer Science Lab at Xerox

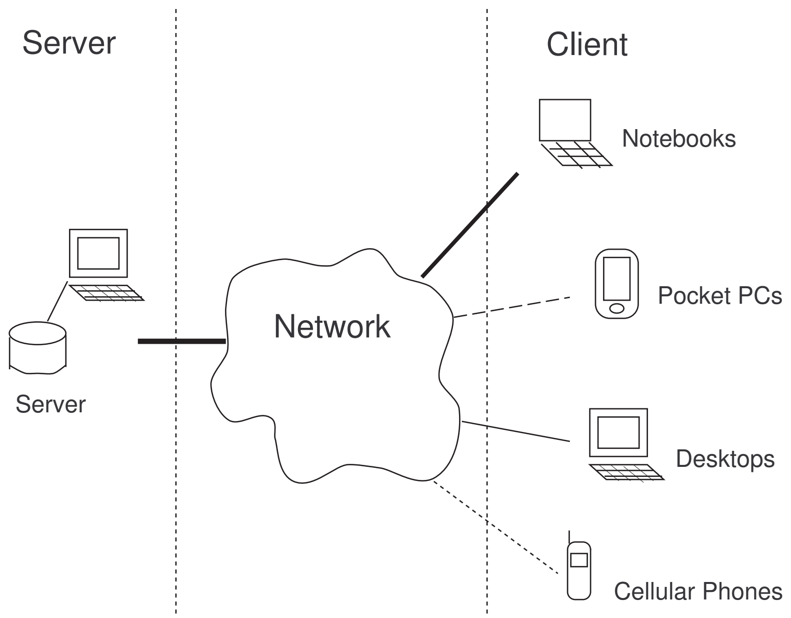

Figure 2.1: System Model

PARC. It has been named as the third wave in computing, a wave that is just beginning. The first wave was *mainframes*, each shared by many users. We are now in the *personal computing* era (the second wave), known as the one-person-one-machine model. Next comes ubiquitous computing, when technology recedes into the background. Ubiquitous computing is roughly the opposite of virtual reality. While virtual reality puts people inside a computer-generated world, ubiquitous computing forces the computer to live out in the world with people. Ubiquitous computing is an integration of human factors, computer science, engineering, and social sciences [53].

The initial incarnation of ubiquitous computing was in the form of "tabs", "pads", and "boards" built at Xerox PARC from 1988 to 1994. Tabs and Boards are now commercial products [53]. Ubiquitous computing involves a larger amount of effort on embedded systems with fewer user-machine interactions, while mobile computing mainly addresses user support. Ubiquitous computing has helped start the recent boom in mobile computing research, although it is not the same thing as mobile computing, neither a superset nor a subset.

Distributed Computing In contrast to centralized computing, in distributed computing the processing is divided into interdependent processes running in computers that are located in geographically different places. The computers in a distributed computing system are usually static elements connected by broadband wired lines. A computer may connect to the system using a wireless link in some special circumstances, but mobility is not the purpose of such a distributed system.

Although a distributed system places the computing elements in different locations, at the application level we want location transparency; we do not want to have to be constantly specifying device location. At odds with this is the idea of agents which seems to put an emphasis on location dependence. Location should make a difference only at the implementation stage. In spite

Table 2.1: Examples of Mobile Devices

Attributes	**Notebooks**	**Pocket PC**	**Handheld PC**	**Cellular Phones**
An Example	Sony PCG-GR170K	iPaq 3670	Stylistic 3500R	Motorola T193
CPU	PIII 1GHz	StronARM 206MHz	Celeron 500MHz	None
RAM	256/512MB+	64MB+	128MB+	-
Display Type	Colour (SXGA+ TFT)	Colour (Touch Screen)	Colour (Reflective SVGA TFT)	Black and White
Display Size (max)	15.1 inches	4 inches	10.4 inches	1.6 inches
Disk Capacity	20GB+	None	15GB	None
Battery Life	2-3 hrs	10 hrs	-	14 hrs

of location-aware systems such as GUIDE [11], mobile applications typically want location transparency.

2.2 Mobile Devices

Over the last decade, advances in digital electronics have made computers smaller, cheaper, and faster. This trend, along with other industry advances, has promoted the development and rapid market growth of small computers that can be carried from place to place.

Table 2.1 shows the current capabilities of some of these mobile devices. These vary in quite a few aspects, from memory size to processing speed to battery life. Mobile applications provide convenient services to mobile users while away from their home location.

Popular mobile devices include the BlackBerry produced by RIM and the Palm handheld produced by 3Com. This research will focus on one particular type of Pocket PC – the iPaq, produced by Compaq.

Table 2.2: Representative Communication Bandwidths

Communication Technology	Bandwidth
Cellular telephone	9.6 kbps
Modem	28.8 - 57.6 kbps
Infrared	19.2 kbps - 4 Mbps
ISDN	128 kbps
Radio	300 kbps - 10 Mbps
Ethernet	10-100 Mbps
FDDI	100 Mbps
ATM	155 Mbps
CDPD	19.2 kbps
Ricochet	128 kbps
GRPS	40 - 115 kbps
EDGE	384 kpbs
CDMA and WCDMA	144 kbps - 2 Mbps

2.3 Current Communication Technologies

As mentioned, mobile computing does not need to involve wireless technology, although wireless is one important form of support for mobile computing. Table 2.2 shows the representative bandwidths for different communication technologies, and shows that there are huge variations in data rates. Bandwidth is only one of the characteristics, however, that varies considerably from one technology to another.

Mobile devices may be connected to the network using a wired link. Current wired link connections have bandwidths ranging from 28.8 - 57.6 kbps using a modem to 10-100 Mbps using Ethernet to faster than 100 Mbps using FDDI (Fibre Distributed Data Interface) to 155 Mbps using ATM (Asynchronous Transfer Mode).

Mobile devices may also depend on wireless technology to make the connection between the mobile device and the server. Current WAN (Wide Area Network) wireless technologies include cellular data with bandwidth of 9.6 kbps or 144 kbps, and CDPD (Cellular Digital Packet Data) with bandwidth of 19.2 kbps. Emerging WAN wireless technologies include Ricochet with bandwidth of 128 kbps, GPRS (General Packet Radio System) with bandwidth of 40 - 115 kbps, EDGE (Enhanced

Data for GSM[1] Evolution) with bandwidth of 384 kbps, CDMA 2000 (Code Division Multiple Access 2000) and WCDMA (Wideband CDMA) with bandwidth of 144 kbps - 2 Mbps.

Another technology in WANs is the WAP (Wireless Application Protocol), an open, global specification developed by the WAP forum [50]. WAP is a communication protocol and application environment designed to support mobile users with wireless devices to easily access and interact with information and services instantly. It can work in different wireless environments such as CDPD, CDMA, GSM, TDMA and GRPS. It also can be built on any operating system, including Palm OS® and Windows CE™.

Wireless LAN technology using the IEEE802.11b standard has a transfer rate of 11 Mbps. Current technology can support IEEE802.11b in a range of up to 100 metres [49]. Two of the popular wireless LAN systems are ORiNOCO™ by Lucent and Aironet® by Cisco.

The most mature PAN (Personal Area Network) wireless technology is BlueTooth [7]. BlueTooth is based on radio technology and supports short-range communications between portable and/or fixed electronic devices to an approximate range of 20 metres. It can support an asynchronous data channel with bandwidth up to 723.2 kbps.

Other dynamic characteristics may affect communication performance, such as the error rate, connection setup time, transmission charges, the density of users, etc. The important consequences of this range of characteristics are: (1) different protocols may need to be employed; (2) changes in application behaviour may be desirable; and (3) changes in application functionality may occur [44].

[1] GSM stands for Global System for Mobile Communication, a wireless technology widely deployed outside North America.

2.4 Mobile Applications

Compared to the steady development of notebook devices, handheld devices such as iPaqs had a slow start since their appearance in the market in the mid to late 1990s. The main reason is the lack of "killer applications" to attract mobile users, but the lack of the "killer applications" was due to the poor capability of the devices and the connectivity available at that time.

The recent rapid improvement of the computational ability of mobile devices and wireless technologies has brought mobile computing into a new generation. The ability of wireless network connections to access the Internet, for example, makes small mobile devices such as pocket PCs far more versatile. Increased connection bandwidth can support many applications, including a new group of mobile applications such as video telephony, on-line conferencing, on-line education systems, on-site detection systems, Internet radio/MP3, universal inbox (email, voice-mail, efax, video-mail), mobile office etc. This group of emerging mobile applications can take advantage not only of a network connection but also of multimedia data. These "killer applications" on pocket PCs and handheld PCs might attract more mobile users in the following ways:

- Pocket PCs can be used as an interface to WAN and LAN applications, such as Web, Internet radio/MP3, cellular phone/video phone, universal inbox, mobile office and thin client applications.
- In a PAN environment, pocket PCs become a user interface for a group of networked devices, such as GPS (Global Positioning Services), cellular phone, printer, barcode scanner, and camera.

The following research ideas [3] on possible pocket PC applications can take advantage of current technologies:

- Presentation Tool: The pocket PC stores all presentation material, presentation is transmitted to a wireless projector, and the pocket PC controls presentation and displays speaker's notes.

- M-Commerce (Mobile Commerce): The pocket PCs act as vending machines, or are used in parking management, bill payment, financial information, and stocks.

- Location-based Services: The pocket PCs are used in GPS tracking, dispatching, or storing location-based information (street maps, weather, restaurants, services, etc.)

- Location-based Operation: The pocket PC presents a user-interface based on location (e.g., in the office, presents task lists for actions performed in the office). It acts as a personal agent by filtering information based on location (e.g., filtering email, phone calls, advertising).

- Mobile Healthcare Professionals: The pocket PC can be used as mobile office accessing information such as patient records, schedules and maps, or used in personal area network such as Stethoscope and barcode scanner.

- Entertainment: The pocket PCs can be used to receive Internet radio/MP3s, pocket TVs, on-line movies, etc.

- Mobile Office: The pocket PC provides software that can keep users organized and productive on the road such as organizer, voice recorder, email, voicemail, efax, and Microsoft Office softwares including Word™, Excel™ and Powerpoint™.

- Security: The pocket PC can be used by security guards. For example, while carrying a pocket PC on rounds, maps can show the location of the alarm and display still or video images of the alarm area. The pocket PC can also be used in home security such as to view sensor information while on vacation.

Chapter 3

Review of Related Research

The following three questions are answered in this chapter:

- What are the requirements and constraints of mobile computing?
- Why do people need mobile applications to be adaptive?
- What has been done in making mobile applications adaptive and what has not?

In Section 3.1, the requirements of mobile users, both functional requirements and non-functional requirements, are stated. In Section 3.2, the main constraints of mobile computing compared to static computing are identified. The problems arising from conflicts between requirements and constraints, and possible solutions, are discussed in the next section. Section 3.4 and Section 3.5 review and classify the research relating to adaptation and awareness-based applications. The last section reviews example systems related to this research.

3.1 Requirements of Mobile Users

Mobile users have both functional and non-functional requirements. Functional requirements address the basic services that applications provide while non-functional requirements address QoS (Quality of Service), including issues such as performance or availability.

Functional requirements include:

- Network connectivity. For some applications, such as file service, continuous connectivity is not required since disconnections can be managed with the help of hoarding and caching [22]. For other applications, such as FTP, disconnection can mean the failure of functionality.

- Display ability. Some mobile devices, such as notebooks with 15.1 inch TFT displays, can receive full 256 resolution image, for instance, Web pages, while other mobile devices, such as cellular phones with 2.0 inch monochrome displays, cannot support any colour service at all. Some mobile devices, such as new models of iPaqs, can support colour service but are limited to 4.0 inches in display size.

- Memory requirements. Some applications need a certain amount of memory to run and a shortage of memory can mean failure. For example, to run a video play program in Java, the JMF (Java Multimedia Framework) must be installed. Such a program cannot run on current iPaqs because their limited memory cannot support JMF. Another example could be that some applet programs are written without concern for memory size and this will cause the applets to fail on a memory-limited device.

- Battery power. Battery power is a fundamental requirement for mobility. Some applications are more expensive in terms of energy consumption, while others use less CPU time. Devices that have larger memory consume more energy than similar devices with smaller memory, since keeping more memory alive while in mobile mode requires more energy.

Non-functional requirements include:

- Seamless mobility. Mobile users wish to use their home data and service from any location at any time using any mobile devices, without extra setup [22]. Meeting this requirement needs support from various layers in the network, including wireless technologies in the physical layer, improved routing algorithms in the network layer and possibly different adaptive applications in the application layer.

- Reasonable "key to glass" response time in Web applications. "Key to glass" response time refers to the time that passes after a user clicks the mouse button or presses the return key to submit a request to a Web server to the display of the results of the request on the receiving monitor [12].
- Mobile users may be willing to sacrifice some performance for mobility, but only some [8]. For applications involving video and/or audio, such as network conferencing, on-demand movies, or on-line education, performance requirements include the fidelity, the gray scale, the smoothing between frames (or jitter), and the size of the viewable display. There will be a spectrum of choices for each. At one end of this spectrum will be the best quality of service that requires the best network conditions. At the other end of this spectrum will be the worst quality of service that requires the least network conditions. Somewhere in between will be a break point that the user can tolerate. When performance is below this point, users stop using that service.
- Different mobile user preferences. Mobile users may have different preferences in the level of detail or some other parameters while accessing a particular data provider [35]. For example, to find out how to arrive at an interesting place using an e-map, those tourists who prefer to take the bus want information on bus routes and bus schedules, whereas those tourists who will drive want information on roads and nearby landmarks.

3.2 Constraints of Mobile Computing

Mobile users operate in constrained environments. Fundamental constraints of mobility include [42]:

- Mobile devices are resource-poor relative to static devices. These resources include processor speed, memory size, disk capacity and network connection speed and the ability to handle different data formats. In the case of notebooks, there are some models (called desktop replacements) that have quite similar settings to the best desktop models at that time frame, with the fastest CPU, largest RAM and large hard drive. But since the price of these models is always 2-3 times more than a similar desktop, the most popular notebooks are still relatively poorly resourced. Other mobile devices are resource-poor to reduce their size and weight, considerations that make the mobile devices suitable in different mobile environments. The limitation on the ability to handle different data formats comes from either one of or a combination of the hardware, the operating system and the application. Some mobile devices, for instance, a smart phone, can support only voice, although the incoming data could be from a Web site. A Web-enabled cellular phone could support voice, text and even images, but could not support colour images. The operating system on a mobile device may not provide a built-in decoder for PostScript encoded text, for example. An other example is that Philips currently has an MPEG-4 (Moving Picture Experts Group version 4) player called WebCine for Windows 9x and up, but does not have a pocket PC version of the player.

- Because mobile devices are mobile, they are inherently more vulnerable to theft, loss, or damage. This is less of a problem when mobile devices are used as end users of services with minimal requirement for long term data storage. When the mobile devices are lost or damaged, users will not experience extreme pain over the loss of their data.

- Mobile connections are highly variable in both performance and reliability, mainly due to the limitations of current wireless technology.

- Mobile elements rely on a finite power source. For a mobile device without a hard drive, running out of battery means the loss of all the data on it.

3.3 Challenges and Solutions

The challenges of mobile computing arise from the conflict between two things: mobile requirements from users and applications; and mobile constraints from the network and the mobile devices.

From the client's point of view, this includes how to reach the remote server, how to maintain a stable connection, and how to manipulate (play, decompress and/or keep) the video/audio data received. From the server's point of view, the challenges include how to deliver data over challenging network conditions, and how to accommodate heterogeneous clients and deliver data in formats suitable for the devices that need it. There are also other challenges presented by mobility, such as challenges relating to billing and security, which are not addressed in this project.

Solutions can be found either by improving network technology and topology or by improving the hardware technology of the mobile device itself. At the physical layer, improved technology of cellular communications, wireless LAN, and satellite services will increase the bandwidth, reduce the error rate, and switch between different platforms smoothly.

At the network and the transport layers, improved protocols may make it possible to deal with the high link error rates and spurious disconnections related to a wireless network [6]. Another example of adaptation on this layer is the transformer tunnels proposed by Sudame [46]. Transformer tunnels provide route-specific adaptations by selecting a proper packet size, transmission rate, encryption method, etc. Approaches in this layer usually focus on improving TCP (Transmission Control Protocol) throughput and providing services for mobile users to access information anywhere and at anytime without any extra setup, hook up or registration. At the session/presentation layer, a framework was proposed to stream multimedia data over IP and Mobile IP networks to heterogeneous mobile devices in multicast groups [13].

At the application layer, applications can be made smarter by making them adapt dynamically to any changes of the user, of the location, or of the resource. Since the network has been added great complexity while many billions of tiny embedded devices worldwide are having some networking capabilities, it is impossible to achieve any form of static planning or optimization of network operations. The real goal of adaptive networking is to provide robust end-to-end service, where the end points are located in applications. Hence considering the needs of applications and their users and putting adaptation in the application layer are more promising than any approach that addresses the network level alone. Software-supported adaptivity has demonstrated its real power in systems such as Odyssey [38] and TranSend [20, 19].

This research will focus on how mobile adaptive applications can take advantage of resource awareness.

3.4 Review of Related Research in Adaptation

In the application layer, many adaptation strategies can be applied before the content is delivered over networks. There are different ways to classify these adaptation strategies. These classifications are useful in providing a big picture of the adaptation studies in a logical way. For example, these strategies can be classified into one of the following two categories [4]:

Lossless adaptation The data is altered in a lossless way while travelling across links with poor connectivity. Some compression technologies, such as run-length coding and DPCM (Differential Pulse Code Modulation), can reduce data volume without any loss of information [55]. Various systems can provide lossless support without any application involvement.

Lossy adaptation When supporting resource-poor devices such as a pocket PC, lossy adaptation can reduce data volume in order to provide service with reasonable performance, but this means a certain sacrifice in performance. Commonly used strategies include dropping inessential portions of data and using interpolative technology. This approach to adaptation may result in a lower-fidelity presentation, smaller sized display or cartoon-like colour or action.

Adaptation strategies can also be classified based on what kind of information to adapt to:

Adaptation to resource information The resource information considered in this scenario includes the capabilities of the mobile devices and the communication abilities. Systems that have the ability to provide constant service while dealing with heterogeneous devices and connections can fall into this category, such as Odyssey [38] and the Top Gun Wingman browser [18], which converts Web images into black-and-white versions when detecting a Palm handheld as a client. This type of adaptation is studied in this research.

Adaptation to user preference As mentioned earlier in this chapter, users may have different preferences, either in the level of detail or some other parameters.

The need to adapt to user preference is illustrated by the use of Baldwin Piano's Web site [23]. A pianist who is interested in buying a new grand piano will want in-depth information, such as the texture, number and depth of the keys, or of particular models, but a high school student who wants to do a survey on pianos for her economics class is satisfied with general information on all the models, including some pictures and the prices.

Some research on multi-agent systems has addressed this problem, such as Tambe's adaptive agent integration architectures [48]. Adaptation to user preference can provide personalized service, and is one goal of *pervasive computing*, an emerging network concept.

Adaptation to location-based information Adaptation based on the location-based information can provide users with services such as a tour guide [14], which can provide the tourist with information about the scenery nearby while s/he moves around.

According to the adaptation techniques, adaptation strategies can be classified into one of the following two categories [35]:

Static adaptation This is a simple strategy. Different presentations of a data set are pre-processed and pre-stored, differing in quality, data size, or/and processing requirements. According to a request from a client, an appropriate version will be selected and delivered. This strategy works well when adapting to the heterogeneity of the client devices. For example, a resized presentation could be picked for a pocket PC, or a gray-scaled presentation could be picked for a Web-enabled cellular phone. But this strategy fails when experiencing the dynamically changing variables such as network conditions and client memory load. Due to the low cost of storage, and the need to reduce processing overhead at presentation, most current Web sites create multiple versions of data prior to the connection setup time [35].

Dynamic adaptation This type of adaptation deals with the dynamically changing variables during presentation time. The most important variable is the network condition, especially to the video servers. Usually, delivering video data spans a long time period, experiencing changes of the unstable transfer rate. The client CPU load will affect the data format with which a client can deal. The client memory load will affect buffering ability. An example system that supports dynamic adaptation is Odyssey [38], which is discussed in Section 3.6.1.

According to the data types involved in adaptation, adaptation strategies can be classified into one of the following two categories [35]:

Single-type adaptation This form of adaptation switches between different versions or different parameterized data, but still the same media type. For example, data have different layouts, such as either a shorter version or a longer version, but are all text-based. Another example is when the data have different formats, such as either MPEG or JPEG (a standard defined by the Joint Photographic Experts Group [10]), but are all video. Some Web sites, for example, www.micromedia.com, have a regular version for desktop and laptop devices, and a mobile version for small mobile devices such as pocket PCs.

Another strategy is used for streaming video data by some servers. Using this strategy, the data set could be generated in various levels using different parameters such as quality in JPEG and quantization in MPEG. Video data will switch dynamically from one data level to another according to the changing network condition or other dynamic parameters. Some new techniques, such as are realized in the MPEG family [37] after MPEG-4, use more sophisticated object coding algorithms, which could provide stronger support for multi-level switching. Other single-type adaptation strategies include converting GIF images to more compact JPEG [36], and converting MP3 files to AAC (Advanced Audio Coding) files for the same sound quality at approximately 50 percent lower bit rate [16]. The data sets could be pre-stored or dynamically generated, depending on the complexity of the data structure, server storage ability and computing ability of the server. One of the three example applications addressed in Odyssey [38], a video player, adapts by switching between some pre-stored data sets, each of which represents a specific fidelity level.

Cross-type adaptation Using this form of adaptation, a system can convert content into a format more suitable for a specific client device or a different network condition. For example, while delivering text data to a smart phone, a text-to-voice translation will be progressed. Other possible cross-type adaptation strategies includes converting images to a text version explanation when sending data to a Web-enabled cellular phone, and converting video to a still image when experiencing bad conditions for the transfer. Cross-type adaptation can happen either prior to the transmission (i.e., static adaptation) or during the transmission (i.e., dynamic adaptation).

Adaptation strategies can also be classified into one of the following two categories [35]:

Semantic adaptation This type of adaptation manipulates the semantic structure of a Web page at connection setup time by, for example, removing the redundant objects. A new version of the Web page could be generated to improve the efficiency of the data transfer. The redundant objects could be images or video clips of banners, logos or advertisements, which maybe cannot be supported by a client device.

Physical adaptation This type of adaptation delivers physically modified data, involving one of or a combination of conversion, scaling and distillation processes.

3.5 Review of Related Research in Awareness

Since adaptation is based on information relating to change, awareness of change is crucial to building successful adaptive applications.

Generally speaking, awareness can be categorized as resource awareness, context awareness, or user awareness. In this project, resource awareness is the only aspect that will be studied, although the other two types of awareness will be introduced briefly in this section.

3.5.1 Resource Awareness

Resource awareness supports application adaptations triggered by things such as the presence of accessible devices and network capability. The resources include display size, resolution, colour depth, computing power (CPU capacity), storage (RAM capacity), bandwidth, transfer rate (effective bandwidth), network latency, client CPU load and client RAM load, battery capacity, software, operating system name, operating system architecture, and operating system version. In this research, those resources are classified as either static resources or dynamic resources. This corresponds to the static adaptation and dynamic adaptation mentioned in Section 3.4. Adaptation decisions of a mobile adaptive application need knowledge on some or all of these aforementioned resources. To retrieve resource information, extra services are needed by the adaptive applications, including monitoring, calculating and decision making. These services could be provided either by the operating system or by applications themselves. Static information can be collected either by using client profiles or by run-time retrieving. Dynamic information can be gotten only by run-time services. These issues will be studied in detail in this research.

There have been a number of systems supporting adaptive applications, such as Odyssey [38] which incorporated resource awareness. Odyssey's emphasis was not on what resources need to be monitored and how to monitor them, but rather on who is in charge of monitoring the resources. Choi's research [12] addressed resource awareness, but only in terms of a limited set of parameters: type of processor, type of Internet connection, percentage of processor load and percentage of memory load. Acharya's work [1] focused on designing a language, Sumatra, an extension of Java, to support resource-aware mobile programs. It provided interfaces to adaptive applications rather than applications of resource awareness itself that will be emphasized in this research.

Since different applications have different functional and/or non-functional requirements, such as performance or availability, the resources taken into account in this project may be quite different from application to application. How much it is possible to take advantage of the adaptation strategies will depend on how well the information pertaining to those resources is managed.

3.5.2 Context Awareness

Context awareness supports adaptations triggered by things such as the location of the user and the devices around the user. Schilit's study [44] provides a comprehensive discussion of context-aware mobile computing. Lei's work [35] covers a wide range of design issues in context-based media adaptation.

In the fictional scenario presented in Section 2.1, the tour guide can determine when the user's location changes. Actually, this is not fiction anymore since tour guide systems such as GUIDE [14] do exist. Among other applications that can take advantage of context-awareness is a conference assistant system, which can provide information services when a user, who attends the conference, moves to a different exhibition spot or to a different session.

3.5.3 User Awareness

User awareness addresses those adaptations triggered by things such as the coming together of a group of people. User-aware mobile applications support collaboration between mobile users by keeping track of each user's profile. The Mushroom system [32] defines a framework for collaborative working and user interaction on the Internet. Its most significant features are the boundary, which provides security and integrity guarantees for shared objects and communication, and space, in which users are aware of one another while working on shared objects [32]. Some other work is hardware-based, including the IPAD (Inter-Personal Awareness Device) [25], a hand-held or wearable device designed to support awareness and collaboration.

3.6 Review of Example Adaptive Systems

Building an adaptive mobile computing system is not a new idea. Some existing systems and related research are reviewed in this section. Details of various architecture approaches are described in detail in Chapter 4.

3.6.1 Odyssey

Odyssey [38], one of the most important adaptive systems built by Satyanarayanan's group at CMU, focused on resource management for multiple applications running concurrently on the same client machine. As opposed to the server-based design in this research, the adaptation logic of Odyssey was mainly implemented at the client. A new property, *fidelity*, was defined to quantify the notion of quality. By providing a fidelity value, a client can get proper adapted content from the server. The client-based design is not suitable in the target scenario of this research. In this design, the adaptation logic is designed at the server side to reduce the implementation complexity when dealing with heterogeneous mobile devices.

One of the main contributions of Odyssey is that it defined an application-aware approach. By supporting a collaborative partnership between applications and the operating system, the application-aware approach permits individual applications to determine the optimal form of adaptation, but preserves the ability of the operating system to monitor resources and to enforce allocation decisions. The advantages of this approach are discussed in detail in Section 4.1. Due to its superior characteristics compared to other approaches, such as application-transparent and ad hoc approaches that were introduced briefly in Chapter 1, the application-aware approach is applied in this design.

Odyssey supported adaptation based on resource information. It was designed to run in wireless environments characterized by limited and frequently changing transfer rate. To provide better adaptation service to various mobile devices, more resource types, including display size and display type, are studied in this research in addition to the transfer rate.

Odyssey was a well-designed scalable adaptive system that could support general applications. A set of qualitative lessons were learned and the experiments have given a basic view of this research.

3.6.2 An Adaptive Web Server

Choi's [12] work examined how the client computing resources can be used by a Web server to adapt the requested content to a form that best suits the client's machine and thereby decrease the response time.

Four types of resource information were considered in Choi's research: type of processor, type of Internet connection, percentage of processor load and percentage of memory load. The significance of each type was studied. This study shows the importance of different types of resource information to adaptation.

The concepts of *server pull* and *client push* in system design were also defined. In the client push approach, the information accompanies each request from the client to the server. In the server pull approach, information is requested by the server as needed. The trade-offs between these two approaches are discussed in Section 4.4. The server pull approach was implemented in Choi's prototype, a server-based adaptive Web server. Because his work was focused on client computing resources rather than system design, there was no further study of system design alternatives.

3.6.3 TranSend

Fox [20] proposed a proxy-based approach to adaptation, in which proxy agents are placed between clients and servers, but closer to the client side, to perform aggressive computation and storage on the fly. The advantages of proxy-based adaptation are discussed in Section 4.2. This design could be extended to a proxy-based design to gain in different scenarios, which is described in Chapter 9.

A scalable transformational Web proxy, TranSend [20], was developed as an example based on this design. TranSend provides network adaptation for users of slow links and datatype-specific distillation (lossy compression) is applied as the adaptation scheme to achieve better compression when needed. Without the middleware, which is a technology used in this research and will be introduced in Chapter 5, to provide adaptation services, TranSend can not support general applications.

3.6.4 Smiley

Most research on adaptations has addressed the shortage of resources, such as a wireless network that becomes a bottleneck. From another point of view, it is possible that from time to time extra bandwidth is available, especially between two network accesses. An adaptive network prefetch scheme could be used to download files that will very likely be requested in the near future, based on the user access history and the network conditions.

Smiley [30], developed at UCLA, is an intelligent agent that prefetches carefully selected files based on a prefetch scheme which consists of two parts: a prediction module and a threshold module. The prediction module estimates the probability that each file will be requested in the near future. The threshold module computes the prefetch threshold dynamically for each related server dynamically based on system load, capacity and the cost of time and system resources to the user. A formula was derived to calculate the prefetch threshold. Simulation results showed that lower averages of time costs and bandwidth costs can be achieved by prefetching those files whose access probability is greater than or equal to the server's prefetch threshold.

In this design, when extra resources are available, higher quality data will be delivered. Although both Smiley and the system designed in this research are trying to use the extra resources, the main difference is that the client will gain in the future when Smiley predicts correctly whereas the client will gain immediately under this design. There is also potential loss when Smiley fails in prediction.

3.6.5 An Adaptive MP3 Player

As opposed to those strategies that concentrate on adapting the data stream, Kunz [34] proposed another approach on moving partial of the computational effort required at the client to proxy servers. In this work, *object graph* was defined as the composition of objects containing functionalities of a mobile application. There are two object graphs: one resides at the mobile device, and the other resides at the proxy server. These object graphs will be dynamically reconfigured to adapt to any changes in the environment. An example application, an MP3 player, was used to demonstrate his design. Decoding was shipped from a resource-constrained portable device to the proxy, a faster machine. Results show that the performance of mobile applications, especially resource-intensive ones, could be increased and the power consumption on mobile devices could be reduced under certain conditions.

Kunz's work proposed another adaptation strategy using proxies. It can work with the middleware design. One possible extension of the architecture design of this research is to move the middleware on the server to a proxy. The middleware can provide the adaptation service to the adaptive MP3 player and the adaptive MP3 player implements the adaptation scheme as it was originally designed. Moreover, other adaptive applications can be supported at the same time.

3.6.6 Other Examples

Jing, Helal and Elmagarmid [31] wrote a survey of client-server computing in mobile environments. Their survey provides a different perspective regarding the adaptation area which this work addresses. They studied the existing projects in the area up to 1999, the time that the paper was published. In their survey, research was characterized into one of three paradigms: mobile-aware adaptation, extended client-server model, and mobile data access. Mobile-aware adaptation covers the research on the strategies and the techniques of adaptation and operating system support. Some studies of application-transparent adaptation and application-aware adaptation, such as Coda [33], WebExpress [26] and Odyssey [38], were introduced. The extended client-server model includes those studies of extended architectures that support mobile client-server information access. Extended architectures fall into one of three categories: thin client architecture, full client architecture and flexible client-server architecture, which is a middle solution between the former two types of architecture. Mobile data access addresses the problems associated with delivering server data over wireless and mobile networks. Projects on server data dissemination and on client cache management were reviewed. Their survey provides not only pointers to many important projects on mobile computing but also comprehensive studies of these three paradigms.

This research addresses all three paradigms mentioned in their survey. An extended client-server architecture is designed to improve the mobility of the data access, and application-aware adaptation is used. Seven general design issues are discussed. Most of these issues have been addressed by Jing's paper and other papers, but they were mainly introductions rather than discussions in this work. Advantages and disadvantages of each approach, especially the trade-offs between these approaches, are discussed in detail in the next chapter.

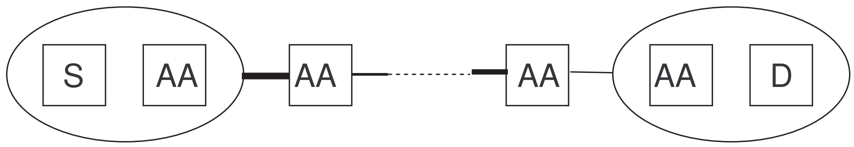

Figure 3.1: High Level Structure of Badrinath's Conceptual Framework

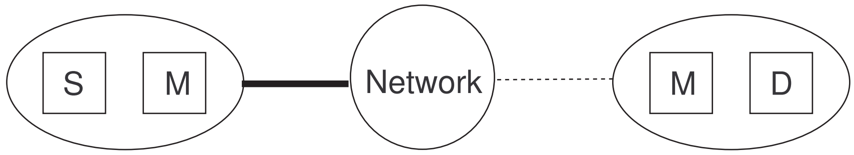

Figure 3.2: Mapping the Design in this Research in Badrinath's Conceptual Framework

Another related work is Badrinath's conceptual framework [4] which summarized the results of several projects in this area, including TranSend [20, 19], Odyssey [38, 39, 40], Conductor [54], Smiley [30], Coda [33] and Rutgers' environment-aware API [46]. His work extracted the general structure of adaptive systems. The architecture design in this research can also be mapped in Badrinath's conceptual framework.

Figure 3.1, a graph borrowed from [4], shows the high level design of Badrinath's conceptual framework. A simple data flow with a single source (S) and destination (D) is transmitted across various links and nodes in the network. *Adaptation Agencies* (AA) represent many different kinds of adaptation mechanisms such as adaptive protocols. All adaptive components are optional, and any AA could be replaced by multiple AAs. All the six network adaptive systems were carefully mapped into this framework, although they may differ in some details. This design can also be described in a high level form and mapped into Badrinath's conceptual framework as shown in Figure 3.2, where a middleware (M), substituting the AA, is used to provide adaptation services such as monitoring the resources.

Chapter 4

Design Issues

Chapter 3 addressed the necessity of adaptation in mobile applications and the need for resource-awareness services for adaptive applications. The following questions arise: How should such adaptive mobile applications be built? Is operating system level support needed to provide adaptation service to applications? How does the operating system monitor resources? How do the operating system and the applications divide their responsibilities? How can the adaptation strategies be applied? And where should they reside (at the server side, at the client side or at a proxy)?

Due to the wide variety of mobile applications, the answer to these questions is always "it depends." while there are advantages and disadvantages to each solution, there is always the question of towards which end of the spectrum of possible choices a particular approach will lean. These considerations and the questions above will be covered in this chapter. In this chapter, seven design issues related to building a resource-aware adaptive mobile application are discussed in detail with a focus on resource awareness, adaptation, and system design from a networking point of view rather than from a software engineering point of view.

4.1 Adaptation Approach: Application-transparent versus *Ad Hoc*

In a modern computing environment, the more functionality the applications have, the more complex the systems to support the applications could be. Advanced

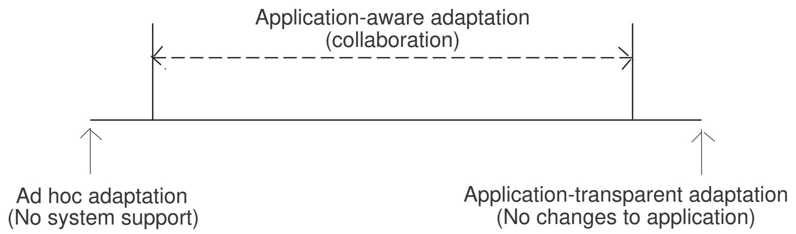

Figure 4.1: Range of Adaptation Strategies

functionality can include adaptation strategies added onto an existing application, monitoring modules added onto an existing system, and the application of new, modified, or improved protocols. Reducing the complexity of the systems while maintaining advanced functionality involves some system design issues. The first issue is who takes responsibility for the adaptation.

Figure 4.1 is borrowed from [31]. The range of approaches for adaptation is delimited by two extremes. At one extreme, *ad hoc* adaptation, adaptation is entirely the responsibility of individual applications. Although an *ad hoc* approach requires no support from the operating system, it lacks a central controller to resolve incompatible resource demands of different applications and to enforce limits on resource usage. It also makes applications more difficult to write, and fails to amortize the development cost of support for adaptation.

At the other extreme, application-transparent adaptation, the responsibility for adaptation is borne entirely by the operating system. This approach is appealing because it is backward-compatible with existing applications; they continue to work in a mobile environment without any modifications. The operating system provides the focal point for resource arbitration and control. The drawback of this approach is that there may be situations in which the adaptation performed by the operating system is inadequate or even counterproductive for some applications. This approach also may need some manual intervention by the user, and therefore, it may not be agile enough to react to changes in the environment. Coda [27] is an example in which adaptation is done entirely by the operating system.

Between these two extremes lies a spectrum of possibilities that are collectively referred to as application-aware adaptations. By supporting a collaborative partnership between applications and the operating system, this approach permits individual applications to determine how to adapt, but preserves the ability of the operating system to monitor resources, to notify applications with resource information and to enforce allocation decisions. Odyssey [38] is designed to gain control from both the operating system and the applications. An approach that falls in this category aims to overcome the shortcomings that come from either an *ad hoc* approach or an application-transparent approach. Certain application-specific conditions may influence the developer to choose an approach closer to one end of the spectrum or the other.

4.2 Location of Adaptation Logic

Adaptation logic can be located at the server, at the client or at the proxy, yielding server-based adaptation, client-based adaptation, or proxy-based adaptation, respectively. Any combination of the above, for example, client-server adaptation, is also possible [31].

Some previous designs achieved adaptability by modifying client applications such as Web browsers or media players. This client-based adaptation approach does not scale well when there can be many different implementations of a particular application. For a single video server, for example, a user may prefer to use Microsoft Media Player, Real Player or some other proprietary or open source player. Even for players from the same vendor, different versions will be used on different mobile devices. For example, Pocket Divx^{TM} (Windows CE version) will be used only on pocket PCs, whereas Playa (Windows 9x/2000^{TM} version) will be used only on other Windows operating systems. An architecture that requires modifications to client applications will increase the complexity of the task dramatically. On the other hand, modification of the server is required only once, and the server is usually configured and maintained by professional system administrators rather than by end users.

A proxy can provide three types of services for data transport: 1) transcoding (filtering and compressing the data stream to suit the current wireless bandwidth [34]), 2) computing/storing, and 3) standardized protocols (e.g., WAP). WAP negotiates device capacities in terms of content characteristics and uses WML (Wireless Markup Language) to adapt by filtering the HTML (HyperText Markup Language) Web content.

Using a proxy at the client side can hide variations in the device capacities and changing/limited wireless network conditions, and make adaptation transparent to both the server and the client. Bandwidth demands can be reduced by using caching and transcoding. The ability of incremental deployment and rapid prototyping is improved [20] by applying proxies in the model. By offloading some parts of the computational load to a proxy, performance of applications is increased and the power consumption of mobile devices is decreased by reducing the CPU cycles and the memory needed to achieve certain tasks [34].

At the expense of such resources at the proxy, this limits scalability. A server-based approach can overcome the shortcomings of the client-based approach, such as being costly to implement, and retain most of the advantages of a proxy-based approach, such as being easy to manage and maintain. Server-based adaptation is difficult to support, however, on non-standard mobile devices. It is hard for a server to have the knowledge of the settings of each particular mobile device at development time and be ready to deal with them, but a proxy-based approach and client-based approach can provide better service in this case.

4.3 General Adaptation versus Application-specific Adaptation

The third system design choice discussed here is whether to provide control over a collection of unrelated applications or support only a narrowly-defined class of applications (in the limit, only one). Odyssey [38] is an example of the first approach and TranSend [20] is an example of the second.

From the user's point of view, however, there is no substantial difference between these two types of adaptation. In both cases, the applications will present the same interface to users.

From the programmer's point of view, using one system to support unrelated applications can result in an overly-complicated system. Furthermore, adding one more unrelated application into the system may be disastrous since contradictory requirements from different applications may cause the system to malfunction. This may involve system redesign, partial or complete recoding, or reconfigurations. It also means a low degree of generalization since some of the attributes of a subgroup of applications may not be the general attributes of all the applications. On the other hand, using one system for each application provides the most subtle support, but ends up separating different applications and imposing a heavy workload on lower layer support in the network.

The new challenge will be to determine which applications can be grouped together (or the degree of the modularization) to achieve the best system design. Some considerations include if they have common or similar resource requirements (crucial point) and where the location of the adaptation logic is. For example, if the adaptation logic is located at a proxy, more applications could be grouped. Fewer applications could be grouped if the adaptation logic is located at a single server machine.

4.4 Server Pull versus Client Push

Two types of data, content data and resource information, will be delivered over the network when adaptive service is provided. There are two approaches to getting resource information from the client to the server according to who initiates the request. In the client push approach, the transfer is initiated by the client and the information accompanies the content request from the client to a server. In the server pull approach, information is requested by the server as needed [12]. Upon receipt of the client request, the server is responsible for analyzing the client computing resource information to determine an appropriate descriptor to be applied to the presentation of the content being requested by the client.

The choice of whether to pull resource information or push it is based on the implementation complexity. The server pull approach is chosen for this project because implementing the client push approach would require modifying a Web browser or a media player which may consist of millions of lines of code. Pull is also more flexible than push in regards to the types of resource information needed by the server application. On the other hand, the main advantage of the client push approach is that the server does not need to ask for the resource information, and so, at least one round trip is saved.

4.5 Acquiring Information: On-demand Monitoring versus Continuous Monitoring

Resource-aware systems must be able to determine the availability and quality of the resources in their environment. To determine this information about the resources, additional messages need to be transferred over the network. Although more frequent monitoring can bring more useful information to the application, this must be balanced by the extra pressure on some bandwidth, memory, or disk space, and CPU time. A resource can be monitored in either on-demand or continuously.

On-demand monitoring is suitable in the following three situations. First, the resource is expensive and is infrequently used. For example, an application uses a 28.8 kpbs modem to periodically check incoming mail being held at a mail server. Second, the quantity of resource is changed infrequently. For example, an application makes some decision based on the amount of disk space available at remote machines. Third, the resource is expensive to monitor and the cost of monitoring is greater than the potential gains. For example, an application transfers a large file over a very low bandwidth link [1]. In other cases, it would be possible or feasible to perform continuous monitoring.

The request for information and the delivery of that information can be either synchronous or asynchronous. A synchronous request immediately returns information about the current availability of the resource in question, whereas an asynchronous request would monitor the resource and notify the application only when the resource availability no longer satisfies an application-specified predicate [1]. On-demand monitoring must be synchronous. For example, an application transfers a sequence of large files and checks the condition of the network before starting each transmission. On the other hand, continuous monitoring can be synchronous or asynchronous. For example, while streaming video, asynchronous return is preferred when the monitoring component is continuously checking the network [1].

For continuous monitoring, an important decision is the granularity at which resource changes are to be monitored. One extreme is to report every change to the requesting application, which is impractical since most applications can tolerate a certain level of jitter (i.e., variation in the changes) in most cases. A commonly used scheme is to use a jitter threshold and track only those changes that are larger than this threshold. Several researchers have proposed jitter-threshold-based schemes as a way of dealing with resource variability on mobile platforms [5, 39]. These schemes work well except that unnecessary reports will be generated when the changes are spiky. Therefore, a function which allows the registration of application-specific (or resource-specific) filters needs to be added to the application so that the application can determine what changes should be reported.

4.6 Centralized Monitoring versus Distributed Monitoring

For different applications, different resource constraints are likely to govern the decision to use separated monitors or not – e.g., network latency, network bandwidth, server load, CPU cycles, etc. The decision is between using a centralized resource monitor (i.e., a single monitor for all resources) and using a distributed monitor (i.e., each resource is monitored by a separate monitor). A centralized monitor facilitates applications that might need information about multiple resources. It also reduces communication requirements for distributed monitoring since information about multiple resources can be sent in the same message. If only a few of the resources change, a distributed monitor can often save the time of collecting all the information for all the resources and save bandwidth as a result.

A combined approach is also possible. A centralized monitor for static information such as the display size and display type can be created for each client. Information will be retrieved after the server gets the request and before it sends the content to the client. For those applications that need dynamic information, such as the instantaneous transfer rate and CPU load, those resources will be monitored separately by different monitors during the rest of the application activity, which should provide better performance than only one type of monitoring.

4.7 Level of Abstraction

In the case of using operating system support for resource-aware adaptation services, such as the middleware designed in this research, there is a design choice on whether to send raw data about the resources or to send processed information to the applications. This choice is heavily influenced by design decisions previously discussed. In the following discussion, a middleware approach is assumed for providing adaptation services.

When an application-transparent approach is used, the operating system will process and use the information itself and the application will not notice the heterogeneity of the clients and the changes in the network. At the other extreme, in an *ad hoc* environment, the raw resource information will be obtained and processed by the individual application based on its own requirements. When using an application-aware approach, the middleware will send either raw data or processed information to the application to increase performance; but the degree of the information processing will depend on the type of the applications supported.

If the middleware is designed to support general applications, the resource information that the middleware obtains will be sent to different applications in raw form, and such middleware may have its own interpretation of the information.

The more specific the middleware is to be, the more the middleware can work on the information, more detailed resource information will be provided to application and finer grained adaptation service can be achieved. The information processing considerations include:

- Some resource information can be used in its raw mode directly, such as the display size and display type. Applications will decide the proper media data to transfer based on specific size and type.

- Some information needs to be grouped into information more convenient for decision making, such as transfer rate. Transfer rates can vary widely and continuously from 0 bps (disconnected) to 100 Mbps (broadband). It is impossible to specify adaptive data for each transfer rate. It is sufficient to divide the range into several levels: the more levels, the finer the adaptation required, but the overhead will be increased. The number of levels is also related to the frequency of changes that the user can tolerate. Dynamic adaptation means providing more elaborate service but also means changing the data during the communication; the finer the adaptation, the more frequent the changes. Too frequent changes are usually annoying. Picking a proper balance point for an application is a difficult design decision, and sometimes the choice could involve a user decision. Odyssey [38] provides a service that allows a user to make the decision as to whether or not s/he is willing to change. A more intelligent system could learn the behaviour of certain users and make the decision itself.

4.8 Summary of Design Issues

The first three issues discussed in this chapter deal with operating system support. These issues are closely related. For example, application-transparency versus *ad hoc* is related to server-based adaptation versus client-based adaptation versus proxy-based adaptation in that all the three adaptation types can happen only while using an application-aware or an application-transparent adaptation approach.

The other four issues are from a higher policy point of view. "Server pull versus client push" addresses the problem of who should initiate the resource information request. The next two issues address how to monitor the resources. The last issue addresses the level of the resource information that should be provided to application when using an application-aware approach to adaptation.

Chapter 5

Designing the Adaptation Architecture

The literature review and the discussion of design issue leads to a new architecture design for heterogeneous mobile environments. This architecture provides server-based adaptation using middleware that can monitor resource information of mobile devices and notify multiple adaptive server applications. This does not mean that all the server applications should necessarily exist in the same server machine, but each application could be attached to the middleware to gain resource-awareness support.

The design considerations of the architecture will be described in detail in Section 5.1. Section 5.2 describes the design of the middleware. Resource information will be addressed in Section 5.3. A new protocol is needed in such an architecture to obtain the static information required between the new peer middleware modules, at the server and the client. This protocol defines the resource types of interest, which are used as the keys to retrieve the corresponding information. For dynamic information, automatic notification is required. A mechanism to provide applications automatic notification services will be introduced in this section. Modifications to the existing server applications and new interfaces are described in Section 5.4. The event flow, described in Section 5.5, will be helpful to show the complete transaction procedure.

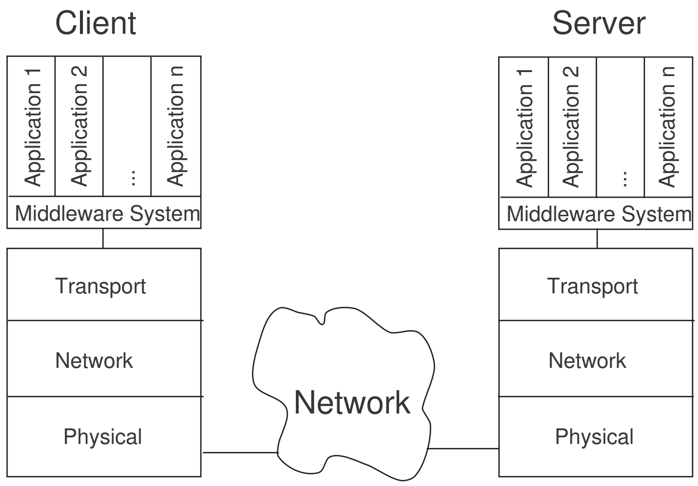

Figure 5.1: System Architecture

5.1 General Issues

5.1.1 Objectives

The problem to be addressed is how to support multiple server applications providing adaptation services while dealing with heterogeneous mobile devices and changing resource conditions. The solution should be simple, extensible, and general, and if possible, it should be transparent to client devices.

5.1.2 Extending the Client-Server Model

The architecture presented here to support adaptive mobile applications is an extension of a classic client-server model [31] in a standard network architecture. In a client-server model, a server acts as an information centre, and a client sends data requests to the server and receives data over the network. The classic client-server model assumes that there is no change in client location and network conditions and that client capacities are similar. These assumptions are not true in mobile environments today. The mobile environment contains heterogeneous mobile devices and varying network conditions due to wireless communications. Extending the classic client-server model is one solution to support mobile computing, as shown in Figure 5.1, where middleware is added to the application layer. Middleware modules exist on both the server and the client.

One of the main features of this architecture is: No modification is needed to client applications, such as a player or a browser. A small program of the middleware runs on the client device, acting as a server to provide the resource information of the client devices. Another main feature is that the existing adaptive mobile server applications which act as the data centre need to be modified to work with the middleware.

This design incorporates two main decisions: using server-based adaptation and applying middleware technology.

5.1.3 Why Server-based Adaptation?

By setting the adaptation logic at the server only, multiple applications can be served concurrently and, moreover, the system can be easily deployed, managed and extended.

As discussed in the previous chapter, using server-based adaptation and leaving the client applications untouched have the advantage of reducing the complexity of implementation and maintenance. Compared to proxy-based adaptation, server-based adaptation can support concurrent applications easily. It also can save more bandwidth.

5.1.4 Why Middleware?

This architecture is designed to support heterogeneous client devices under changing resource conditions such as changing transfer rate. Applying middleware makes the resource information become a service provided by the operating system infrastructure instead of being manipulated by each application itself. Each application needs only to specify to the middleware its own *interest list*, a list of the resource types of interest. This is an application-aware approach.

Middleware overcomes the drawbacks of the *application-transparent* approach. For example, the adaptation performed by the operating system might be insufficient or even counter-productive to a new adaptive mobile application. While the operating system supports multiple applications, under a particular network condition, a Web server may not have to adapt while a video server might want to reduce the frame rate to provide acceptable quality. When dealing with different mobile devices, the same server application might adapt in different ways, for example, by adapting colour resolution to a laptop but adapting display size to a Pocket PC. Therefore, migrating the function of adaptation to applications can provide better service to clients.

This design avoids the high cost of upgrading individual applications using an *ad hoc* approach, since the common services are provided by the middleware. To cooperate with the middleware, the server applications can be upgraded by implementing the interfaces described in Section 5.3 and 6.2, and applying the corresponding adaptation strategies based on the resource information from the middleware. The design of the middleware also makes it possible to leave the client applications unmodified.

5.2 The Middleware

As shown in Figure 5.1, the middleware module exists on both the server and the client. These peer middleware modules will communicate to get resource information from the client device to the server, with the middleware on the client acting as a resource information server.

The middleware module at the server can be installed on the same machine as the server applications, or on a proxy close to the server applications. The middleware module at the client will have to be installed and run in the background on the mobile devices, prior to the request for adaptive service.

There might be different versions of the middleware module, due to variations in hardware and operating system. For example, different mechanisms will be used to retrieve the CPU load information between a Windows-based environment and a

Unix-based environment, although the interface should be the same from the mobile application's point of view.

To provide better adaptation service, as discussed in Chapter 4, the main function of the middleware will be monitoring the resource information and notifying the server applications. The adaptation will be left to the server applications.

The functions of the middleware at the server include:

- Communicating with the client and retrieving resource information about the client device.
- Monitoring dynamically changing resources and providing information on changes.
- Processing resource information.
- Providing resource information to adaptive mobile applications.

The functions of the middleware at the client include:

- Collecting resource information.
- Providing resource information.
- Assisting in generating dynamic information reports.

5.3 Resource Information

The main goal of this architecture is to take advantage of the information provided on client capabilities and changing conditions.

The information that the middleware retrieves can be categorized as static or dynamic. Table 5.1 shows a list of the resource types considered in the prototype for this study. The list could be extended endlessly since the relevant information on resources is application-specific and new applications continue to be developed.

The static information represents the characteristics of the client device, including display type, display size, CPU capacity, disk capacity and bandwidth, etc. This

Table 5.1: List of Resource Information in Consideration

Resource	Type
Display Size (in pixels)	Static
Display Type (colour or black_white)	Static
Bandwidth (in kbps)	Static
CPU Capacity (in MHz)	Static
Disk Capacity (in GB)	Static
RAM (in MB)	Static
Round Trip Time (in ms)	Dynamic
Transfer Rate (in bps)	Dynamic
CPU Load (%)	Dynamic
Memory Load (%)	Dynamic

type of information is fixed during one connection. The bandwidth may change when the user changes the connection method, for instance, from CDPD wireless service to wired Ethernet LAN, but generally, the switch causes disconnection from the network. As a result, the re-connection will be considered as another connection in a traditional Internet environment. All the static information is maintained by the middleware on the client device. Each application has its own interest list of static resources. Only information in the interest list will be retrieved and provided to the corresponding application.

The dynamic information represents characteristics which can change while transferring data, including the RTT (Round Trip Time), transfer rate, CPU load, RAM load, etc. The dynamic information will be monitored based on the interest list provided and on sampling rates chosen by the application.

The static information needs to be transferred only once for each connection, while updated dynamic information needs to be sent to the adaptive application by the middleware dynamically. To support static and dynamic information transferring requires a static resource monitoring protocol and a mechanism for dynamic notification.

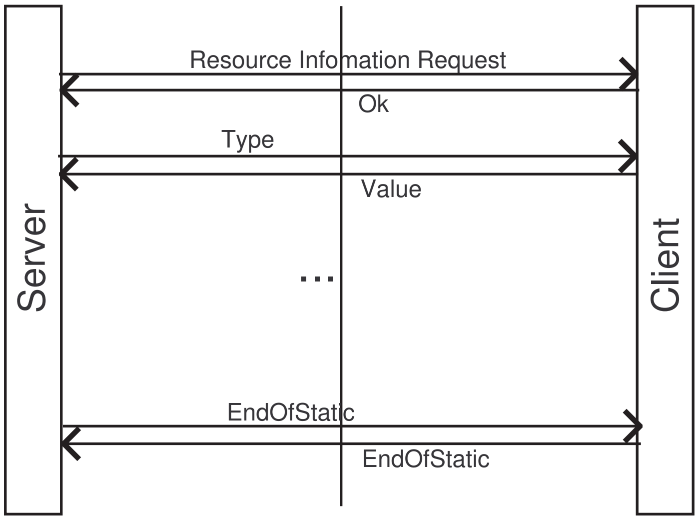

Figure 5.2: Static Resource Monitoring Protocol

5.3.1 A Protocol for Static Resource Monitoring

A new protocol is needed between the new peer middleware modules in the server and the client in order to gain the static information required.

Figure 5.2 presents the new protocol used to achieve the static information. A predefined port number, 8888, will be used. For each connection, the server application will first retrieve static information from the client. Each resource type in the interest list will be picked and sent to the client, and the related value will be retrieved from the client. An "EndOfStatic" will indicate the end of the static information transfer.

5.3.2 A Mechanism for Dynamic Information Notification

While the application is transferring the data, the middleware module will monitor the network status and the dynamic resource status on the client device, and generate an instantaneous report to the server application. Therefore, a mechanism is needed to support dynamic information notification for that application. However, when the architecture is implemented in different languages or the middleware is placed on a different machine from the server applications, different mechanisms must be used. The following describes a mechanism that is suited to the case of implementing in an object-oriented programming language and locating the middleware on the same machine as the server applications.

To support the notification to the application while transferring the data, the middleware must be able to send messages to the application. This ability can be achieved by passing a pointer of the server application to the middleware, referring to the server application as part of the middleware initialization process. There will be a collector in the server application to maintain the dynamic information in the application's interest list. The middleware will regularly get the dynamic information updated in both itself and the server application via the pointer. When it is the time for the server application to see if any adaptation decision should be made, it needs only to check the local collector containing the dynamic information. If the dynamic information has changed significantly, some adaptation strategy will be applied.

5.4 Modifying Existing Server Applications

Since this is a server-based design, the server application is the only place to handle adaptation strategies. To obtain the adaptation services from the middleware, the server application has to be modified. Therefore, adaptive server applications can get and make use of resource information, such as capability of the client devices and the dynamic network and client conditions.

A set of standard interfaces to the middleware is created for the server applications to attach easily. Because the implementation of the middleware is language-dependent, the interfaces will be language-dependent as well. An example of a set of interfaces in Java is proposed in the prototype described in the next chapter.

The applications also need to add adaptation strategies, so that adaptive content can be generated and returned to the client. Some adaptations will be based on static information from the client device – for example, sending an appropriately resized image or a black-and-white image instead of a colour image. Some adaptations will be based on dynamic information – for example, sending lower resolution data or dropping frames when the transfer rate is low. These adaptation strategies are application-specific. Since each application has its own approach to mobile data management, it is better to leave the adaptation to the application rather than to by the middleware.

5.5 Interactions Between the Adaptive Server and the Client

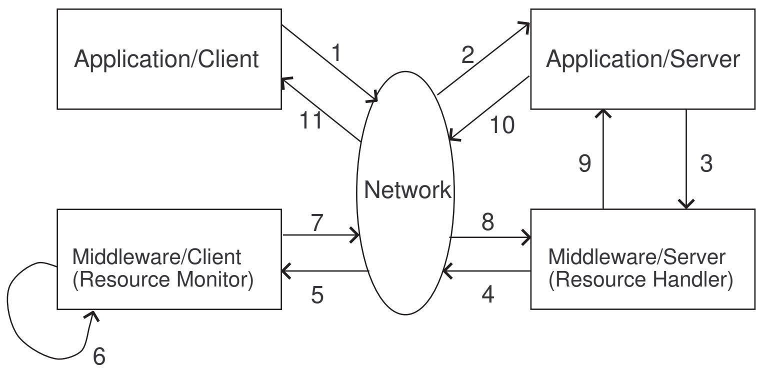

Figure 5.3: Interactions Between the Adaptive Server and the Client

Figure 5.3 shows the interactions between the server and the client in this resource-aware mobile system model.

- (1, 2) The client application sends a request to the server application.
- (3) The server application sends the resource information request to the middleware module at the server side, which is named the Resource Handler.
- (4, 5) The Resource Handler communicates with the Resource Monitor, the middleware module at the client side, using the protocol proposed in Section 5.3.1.
- (6) The Resource Monitor extracts resource information of the client.

- (7, 8) The Resource Monitor sends this resource information to the Resource Handler at the server side.

- (9) The Resource Handler provides the server application with the resource information.

- (10, 11) The server application responds to the client application with adapted content based on the monitored resource information and the appropriate adaptation.

Chapter 6

A Prototype

A prototype has been implemented to demonstrate the model that has been designed. The prototype consists of two components: the middleware and the adaptive server applications. The middleware is used to support resource awareness. The adaptive server applications are existing applications modified to work with the middleware. Three applications, a Web server, a video server and an FTP server, each representing one category of popular network application, are part of this example system.

The implementation assumptions, considerations and decisions are presented in this chapter. The testing of this prototype is described in next chapter.

6.1 Assumptions

The following simplifying assumptions were made in the implementation of the example system.

- Unicast. The communication is based on one client and one server.
- No proxies exist between the server and the client. The client and the server are communicating directly through sockets.
- The server has enough capacity to support clients. Scheduling at the server is not considered in this research.

- Static information will be read from a profile pre-stored on mobile devices rather than retrieved from the actual operating environment.
- Servers are located on the same machine as the middleware.

6.2 Algorithms and Interfaces of the Middleware

Different dynamic information has different methods for retrieval. For example, the dynamic information about the instantaneous computing ability (CPU load) or buffering ability (memory load) should be determined by the middleware on the client device and sent over the network to the middleware on the server. The dynamic information about the instantaneous network condition (RTT or transfer rate) could be retrieved by the middleware on the server alone.

There are many ways to manage the static information of the mobile device. The simplest way is to use a profile to maintain the configurations of the static information. Although simple, this method needs users with knowledge of the setup of the devices.

A transparent way is to let the program retrieve the information when it initiated. This method is feasible using Java™ on a Unix®/Linux™ system, or a Windows 9x and higher operating system. But it can not be fully supported, at least currently, by the Jeode™*Runtime*, an implementation of Sun's PersonalJava™ 1.2 specification that is used in this research to support Java program on a pocket PC. The difficulty is that PersonalJava is a simplified library of Java, and so some classes are not supported. This includes the java.awt.GraphicsEnvironment and java.awt.GraphicsDevice which can retrieve the colour mode and display size in pixels of the display device in use.

Another difficulty comes from the programming language. Java can get per-process and virtual-machine-wide memory information but not system-wide memory information. Thus, it is hard to monitor memory load of the mobile devices using Java.

In this prototype, RTT is used to estimate the instantaneous network condition. The utility "ping" is borrowed. The middleware will call "ping" based on a specified time interval. The average RTT of "ping" will be communicated to the server applications. Because of the limitations of Java and the JRE™ (Java Runtime Environment) on the mobile devices selected, static information is obtained by a pre-stored profile.

Table 6.1: Interfaces

Interface	Description
public AAHandler(String, Vector, *AdaptiveServer*)	Constructor with host name, resource list and pointer to the application as parameters
protected void setInterestedResourceList(Vector)	Reset the interested resource list
protected void setTimeInterval(int)	Set the time interval for monitoring dynamic info.
protected Hashtable getStaticInfo()	Retrieve static resource info.
protected void retrieveDynamicInfo()	Retrieve dynamic resource info.

Table 6.1 shows the interfaces of the middleware. As mentioned in the architecture design, the interfaces are language-dependent. This set of interfaces represents an example in Java.

To use the interfaces defined by the middleware, the application should implement the Java feature Interface *AdaptiveServer*. Therefore, a reference pointer of a server application will be passed to the middleware and be used to notify of changes in the dynamic information. The other four interfaces in this architecture include: one to set the resource list, one to set the sampling rate of the dynamic information one to get static information and one to re-sample the dynamic information.

6.3 Middleware Implementation

6.3.1 Middleware Design Decisions

The middleware is implemented based on the discussion in Chapter 4. Some important design decisions include:

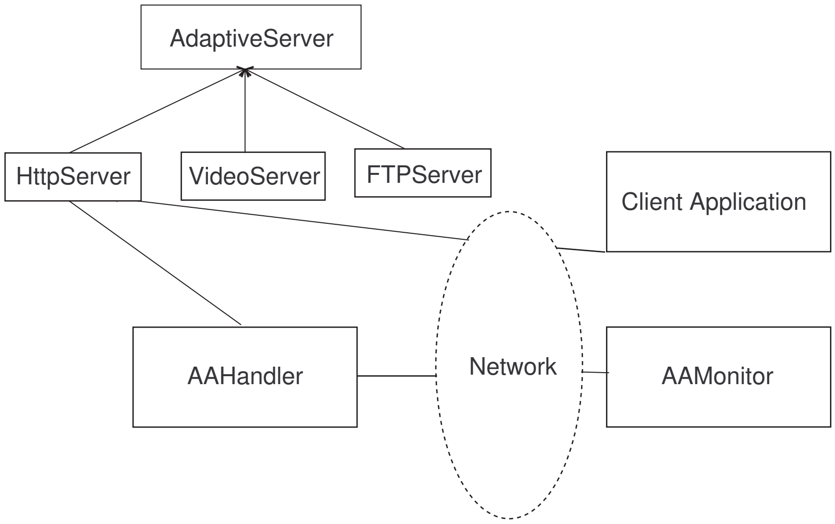

Figure 6.1: Class Diagram of the Middleware

- The server pull approach will be used (i.e., the information is requested by the server when needed) to reduce the implementation complexity.
- Both on-demand and continuous monitoring will be applied. On-demand monitoring happens at the very beginning of a connection and static information will be retrieved. Continuous monitoring is used to obtain the dynamic information when needed.
- Centralization. A single Resource Handler will be used in this implementation, based on the assumption of locating the Resource Handler at the same machine on which the server applications run.
- Generalization. Both the static information and the dynamic information will be reported in raw mode, leaving the application to make decisions based on its individual requirements.

6.3.2 Implementation

There are two main parts of the middleware as shown in Figure 6.1: the AAHandler at the server side and the AAMonitor at the client side. These two parts will

```
display_type colour
display_size 1024*768
bandwidth 10
ram_capacity 64
cpu_capacity 600
```

Figure 6.2: An Example of the Configuration File

communicate using a separate socket to retrieve the resource information.

The AAMonitor is a program running on the client device as a resource information server, waiting for the request from the server and transferring the static information on request using the new protocol described in Section 5.3.1. The AAMonitor should also provide the server with dynamic information such as the instantaneous CPU load.

The static information is pre-stored in a profile, configure.txt, in the working directory of AAMonitor. The resource information is in a Type-Value pair delimited by a tab or spaces. Figure 6.2 shows an example of the file contents. The order of the attributes is not crucial.

The configuration will be loaded into a Java hashtable for fast access when the AAMonitor is started and can be modified by the user based on the device settings. AAmonitor must be restarted after the configuration is modified.

For dynamic information, only the RTT is implemented in this prototype due to the limitations of Java and the capacity of the pocket PC.

AAHandler is a program attached to the adaptive server applications. It communicates with the AAMonitor on the client device and provides resource information of interest to the application. It can also call "ping" to get the average RTT and report to server applications every X seconds, where X is the time interval which can be set by the server application.

6.4 Application Implementation

6.4.1 Application Design Decisions

Three representative server applications, a Web server, a video server and an FTP server, will be developed in this prototype to represent different categories of resource requirements in the context of commonly used applications. Each of these is a client-server application designed to deliver potentially large amounts of data. The performance of other popular applications such as email are much less sensitive to changing resource circumstances. For each of the selected application, the server side will be modified from its original version to fit into the new architecture while the client side will remain the same. That means the server and the client of one application may be written in different programming languages. The resource requirements (i.e., the interest list) are given in Table 6.2.

Table 6.2: Resource Requirements

Type	Resource	Web Server	Video Server	FTP Server
Static	Screen Type	Y	Y	N
	Screen Size	Y	Y	N
	Bandwidth	Y	Y	N
	Disk Capacity	N	Y	N
	RAM	N	Y	N
Dynamic	RTT	Y	Y	N
	Transfer Rate	Y	Y	N
	CPU Load	N	Y	N
	Memory Load	N	Y	N

1. Web Server

   Why selected Web server is a very popular application. The version implemented is based on HTTP (HyperText Transfer Protocol) 1.0. Only image contents or pure-text contents are considered in this simple implementation since different data types will be transferred in a separate connection using HTTP 1.0.

Adaptive strategies Send different versions of a page depending on resources available. For example, the page for an iPaq needs to fit into a 4 inch display with larger font. Under low bandwidth, images will be transferred in an adaptive manner such as lower fidelity or a textual explanation.

2. Video Server

Why selected Adaptation is a key feature of applications which support video data. In this prototype, HTTP 1.0 will be used to deliver pre-stored video clips. Different adaptation strategies are applied to generate different versions of a video clip.

Adaptation strategies Transform compression type, drop frames, use low pass, reduce colour, reduce size and reduce fidelity when transfer rate drops (RTT increases). Increase frame rate, use high pass, increase colour, increase size and increase fidelity when transfer rate increases (RTT drops).

3. FTP Server

Why selected FTP Service is selected to represent those applications that do not care about the length of the transmission time, only that the entire contents of a file must be transferred eventually. There is no particular display requirement on the client device. This application simply bypasses the middleware while running.

Adaptation strategies No adaptation strategy.

6.4.2 Application Implementation – HttpServer

Since both the Web server and the video server are implemented using HTTP 1.0, these two applications are combined in one Java program called HttpServer, which inherits the abstract class *AdaptiveServer* in implementation.

Upon the arrival of a request, the HttpServer will acquire the static information, and an average (recent) RTT to estimate the current network condition of the client via AAHandler before it sends the adaptive data to the client device. The resources in the interest list are application-specific, and are given in Table 6.2.

The Web page sent will be in a version suitable to the heterogeneous client devices. For example, in the case of the pocket PC as the client, the server will send a page with a narrower width (i.e., 240 pixels) and larger font size. In such a format, a user does not need to scroll the display in both dimensions, and the larger font is more comfortable to a user when the user is mobile.

Most Web pages are around 100 to 100,000 bytes [2] and can be transmitted very quickly, even over a slow wireless CDPD connection (100 B / 20 kbps = 0.04 s to 100,000 B / 20 kbps = 40 s). The adaptation based on the dynamic information is not considered in this prototype since it is not likely to impact performance significantly when compared with transferring video data.

For a video server, display type and display size will affect the selection of a resized or gray scaled version. RAM capacity determines how much space can be used for buffering, and the CPU capacity determines if the device has enough computing capacity to handle the decoding process. There are trade-offs between these two resources based on these factors. When the RAM capacity is low, a highly compressed format is preferred. But a high compressed format needs higher CPU capacity to decode. Other solutions have to be managed when both are low, such as decreasing fidelity or cross-type translating. The pocket PC used in this research has sufficient CPU capacity to handle the decoding including divx, a highly effective compression type.

Since this prototype is using HTTP to deliver pre-stored data, dynamic adaptation is not supported at this stage, although the average RTT is retrieved and updated regularly.

6.4.3 Application Implementation – FTPServer

An FTP server based on File Transfer Protocol (RFC 959) has been built in Java. It borrowed the source code by Peter Sorotokin [45] and works well with the vxFtp

[9] for the pocket PC. In this prototype, it has been modified to communicate with AAHandler by inheriting the abstract class *AdaptiveServer*. FTP Server cannot take advantage of the resource information and therefore no adaptation strategy can be applied.

Chapter 7

Testing the Prototype

At the current stage, the prototype contains three modified server applications: a Web server, a video server and an FTP server. The Web server and the video server can transfer adapted data using HTTP 1.0. Both servers pull resource information from the client at the other end of the connection through the middleware. Static information on client resources is collected through the client middleware module over the network prior to transferring the content data. Although dynamic information, such as RRT, can be obtained as described, it is not used in the current implementation.

7.1 Test Devices and Network

The test devices include a desktop computer (Dell Dimension 4100), a notebook computer (Toshiba Satellite 2000) and a pocket PC (Compaq iPaq H3670). These are selected to represent three of the most popular types of client devices. Current desktops have over 2 GHz CPUs, tens of GB of disk space, over 512 MB memory, and displays up to 19 inches. They can be wired with fast network connections, for example, 100 Mbps. Because of the desktops' low price, users can keep their computers at the leading edge of the technology. Notebooks today can be as good as the best desktops in CPU and memory, but are a bit more limited in disk space and display sizes (up to 15 inches). When the notebook is not moving, it can be connected to the same network as the desktop. When the notebook is mobile, the connection to a network is limited by the available wireless communication service. From the family of pocket PCs, the iPaq is picked because it is among the most popular models. The iPaq used has a 206 MHz CPU, which is similar to a Pentium II desktop, 64 MB RAM, a 4 inch colour display, and no hard disk (unless a hard drive PC card is used). The iPaq's mobility and resource limitations cause users to have different expectations while using mobile applications.

The desktop and the notebook are deemed to be resource-rich devices, while the pocket PC is deemed to be a resource-poor device, in CPU, memory, disk space, and display. The desktop is resource-rich device in connection, and the notebook and the pocket PC are resource-poor devices in connection. Example configurations tested are described in Table 7.1.

Table 7.1: Configurations for Devices Used in this Research

Desktop Computer	Notebook Computer	Pocket PC
Dell Dimension 4100 Display Type: colour Display Size: 1024x768 Bandwidth: 100M CPU Capacity: 866M Disk Capacity: 15G RAM: 128M	Toshiba Satellite 2000 Display Type: colour Display Size: 1024x768 Bandwidth: 100M/19.2k CPU Capacity: 850M Disk Capacity: 20G RAM: 128M	Compaq iPaq H3670 Display Type: colour Display Size: 240x320 Bandwidth: 11M/19.2k CPU Capacity: 206M Disk Capacity: 0 RAM: 64M

The connection types include: wireless LAN with bandwidth of 11 Mbps, wired LAN with bandwidth of 100 Mbps, wireless WAN using CDPD with bandwidth of 19.2 kbps.

7.2 Adaptation Strategies and Test Data

The adaptation strategy applied by the Web server in this prototype is using different versions for a text-only Web page: mpeg.htm. There are three versions shown in Table 7.2. The selection scheme is based on the descriptions in the table.

There are many possible adaptation strategies that can be used on video applications such as frame dropping, resizing, compressing/uncompressing, reducing

Table 7.2: Example Web Page Used in the Prototype

File Name	File Size	Description
mpeg_full.htm	233 KB	A full length version for desktop and laptop
mpeg_half.htm	116 KB	A shorter length version for desktop and laptop under poor network connection
mpeg_small.htm	4 KB	A small version for pocket PC

quality, reducing colour resolution, using transcoders and re-quantizing. Due to the complexity and the time limitation, only four strategies are used in this research, and these are discussed in this section.

Frame dropping For video streaming, late frames are useless. If the server still pushes the same amount of data onto the network when the transfer rate is low, the late data will waste the network bandwidth and delay more frames. At this time, the client's display will be frozen. Dropping some frames at the server will not only save bandwidth but will also result in smoother play at the client. This method is also easy to implement since the frames can be dropped without decompression at the server even before being converted from YUV[1] into RGB[2]. Since about 60% of the total processing time is spent converting YUV images into RGB, dropping frames on the fly is a reasonable approach. If too many frames are dropped, the discontinuity at the client may be too obvious. Dropping frames is also limited in terms of the type of frame to drop when using different compression methods. For example, when using MPEG, only B frames can be dropped freely. If an I frame is dropped, the rest of the frames in this group are useless and need to be dropped as well.

Resizing Reducing the image size can reduce the data amount dramatically. Since the iPaq's display size is limited to 240x320 pixels, transferring images larger than this size is superfluous in most cases. Even smaller sized images can be used based on the network conditions. Resizing can be done at the same time as converting the YUV images into RGB, but resizing the viewing dimensions during streaming

[1]YUV: Pixel format captured from the camera and used in television broadcasts.
[2]RGB: Pixel format used in computer-based systems.

is annoying to users. The server application built provides proper sized video to a client based on its display size and network connection type.

Compressing/Uncompressing Compressing can reduce a file size dramatically, and therefore, reduce the bandwidth requirement significantly. With MPEG-2, a compression ratio of 40:1 can be achieved while compressing a video of 780x480 pixels at 30 frames/sec to a stream of 6-8 Mbps. But decompressing needs a certain amount of computing resources, both CPU and memory, on the client device. For those mobile devices with limited computing and storage resources, uncompressing or simple compression such as DPCM scheme is preferred. In this case, streaming is feasible, although bandwidth is not saved, but download-and-play cannot be supported due to the lack of buffer space.

Using different quality levels Reducing the quality level of a video can reduce data size, but also reduces the visual effect to a user. Under some conditions, mobile users may be willing to sacrifice some performance for mobility [8]. While using those mobile applications such as on-line conferencing, and on-line education systems, the resolution of multimedia data is less crucial to a mobile user than it is in entertainment-oriented applications. Therefore, providing multiple quality levels is also a possible adaptation choice.

The example video clip is a 2 minute movie trailer, iceage.avi, for the movie *Ice Age*. Different versions are generated and pre-stored based on a coding scheme shown in Table 7.3. Upon detecting that the mobile device is a pocket PC with display size of 240x320 pixels, CPU capacity of 206 MHz and initial RTT of 2 ms, the version code of 111 is figured out based on the coding scheme. Based on this version code, a file with name iceage111.avi, which is of divx compression format, resized image to fit into the pocket PC's display, full rate, and 100% of original quality, will be delivered. In the description of Table 7.3, ori_size means original image size, rgb means uncompressed format, and "processed by X" means frames have been processed every Xth frame (the rest frames have been dropped). Although Table 7.3 lists all the possible codes based on display size, CPU capacity and RTT, only the files presently used in this prototype are listed in the "File Size" and the

"Description" columns.

Table 7.3: Coding Scheme for Different Versions of the Video Clip

Display Size (pixels)	CPU Capacity (MHz)	RTT (ms)	Code	File Size (MB)	Description
1024*768	$\geq$ 1000	$\leq$ 10	331		
1024*768	$\geq$ 1000	$\leq$ 100	332		
1024*768	$\geq$ 1000	$>$ 100	333		
1024*768	$\geq$ 100	$\leq$ 10	321	11	ori_size/divx/full rate/100%Q
1024*768	$\geq$ 100	$\leq$ 100	322	2	ori_size/divx/full rate/80%Q
1024*768	$\geq$ 100	$>$ 100	323	1	ori_size/divx/full rate/60%Q
1024*768	$<$ 100	$\leq$ 10	311	565	ori_size/rgb/full rate
1024*768	$<$ 100	$\leq$ 100	312	282	ori_size/rgb/processed by 2
1024*768	$<$ 100	$>$ 100	313	188	ori_size/rgb/processed by 3
640*240	$\geq$ 1000	$\leq$ 10	231		
640*240	$\geq$ 1000	$\leq$ 100	232		
640*240	$\geq$ 1000	$>$ 100	233		
640*240	$\geq$ 500	$\leq$ 10	221		
640*240	$\geq$ 500	$\leq$ 100	222		
640*240	$\geq$ 500	$>$ 100	223		
640*240	$<$ 500	$\leq$ 10	211		
640*240	$<$ 500	$\leq$ 100	212		
640*240	$<$ 500	$>$ 100	213		
240*320	$\geq$ 1000	$\leq$ 10	131		
240*320	$\geq$ 1000	$\leq$ 100	132		
240*320	$\geq$ 1000	$>$ 100	133		
240*320	$\geq$ 500	$\leq$ 10	121		
240*320	$\geq$ 500	$\leq$ 100	122		
240*320	$\geq$ 500	$>$ 100	123		
240*320	$<$ 500	$\leq$ 10	111	9	resized/divx/full rate/100%Q
240*320	$<$ 500	$\leq$ 100	112	2	resized/divx/full rate/80%Q
240*320	$<$ 500	$>$ 100	113	1 KB	Web page

7.3 Tests Performed

Five tests are described in this section to show how the prototype works in these five most common cases. The first test uses a notebook as the mobile client to request a page from the Web server through the wireless Ethernet. Figure 7.1 shows the initial state of the AAMonitor, the middleware at the client, with the configurations of the current device. As shown in Figure 7.2, upon request of the mpeg.htm file from the client, the server passes its interest list, including display_type, display_size, and bandwidth, and the 7th line on the screen shows that the AAHandler, the middleware at the server, has done the retrieving of the static information from the AAMonitor. The 8th line shows that the server gets the static information according to its interest list. Next, the server applies the adaptation strategy based on the static information and determines that a full-sized version of the requested file, mpeg_full.htm, will be delivered to the client (see line 12 in Figure 7.2). The "stars" represent the transfer of data, 1 KB each. The communication between the peer middleware modules can also be seen in Figure 7.3, which shows the connection from the server, "cao", and three static information types are sent. The adapted content is displayed in the browser on the notebook, as shown in Figure 7.4.

The second test is to use an iPaq as the mobile client to request a page from the Web server through the wireless Ethernet. Figure 7.5 shows the initial state of the AAMonitor at the iPaq with the configurations of the current mobile device. When the server receives the request of the mpeg.htm file from the browser on iPaq, the server will ask the AAHandler to communicate with the AAMonitor at the iPaq and get the static information of interest (see line 8 in Figure 7.6). In this test, a small-sized version of mpeg.htm, mpeg_small.htm with a size of 4 KB, is delivered (see line 11 and 13 in Figure 7.6). Figure 7.7 shows how the iPaq displays the requested page using Internet Explorer.

The third test uses a notebook as the mobile client to request a movie from the video server through the wireless Ethernet. Figure 7.8 shows that five static information types (i.e., display_type, display_size, ram_capacity, bandwidth and cpu_capacity) are on the interest list upon request of the video iceage.avi (see line 5 in Figure 7.8). Based on the static information and the first RTT (see line 9, 10, and 13 in Figure 7.8), which is 0 in this test, the server applies the proper adaptation strategy, and a file with version code 321 is selected to deliver (see line 15 in Figure 7.8). Figures 7.9 and 7.10 show the final states of the server and the AAMonitor, respectively. After the video file is downloaded, Windows Media Player plays the movie as shown in Figure 7.11.

The fourth test uses an iPaq as the mobile client to request a movie from the video server through the wireless Ethernet. Figure 7.12 shows that based on the static information and the first RTT retrieved from the AAMonitor by AAHandler, the server determines that the video file with version code 111, iceage111.avi, be delivered (see line 15 in Figure 7.12). During the transferring, the RTT might be changed due to the varying of the network conditions as shown in Figures 7.13 and 7.15, with the sampling interval of 1000 ms. The downloading bar in Figure 7.14 tells the progress of the transferring. After being downloaded, the movie is played by PocketDivx on the iPaq as shown in Figure 7.16.

The last test uses an iPaq as the mobile client to request a movie from the video server using the CDPD service. Figure 7.17 shows the final state of the server upon receiving the request. Although the static information is the same as that of Test 4 except for the bandwidth, the RTT is dramatically increased, from 0 ms to 528 ms (see line 11 in Figure 7.17), due to the limited bandwidth of CDPD service. Because the RTT indicates that the transfer rate is too slow to download any version of the movie in a reasonable time, the file selected with version code 113 is in fact an html file to explain the situation and give the user some other choices, such as downloading posters of the movie instead of the video data. Figure 7.18 shows such an alternative.

In conclusion, the tests performed on this prototype show:

- Server applications are modified and work with the middleware.
- The peer middleware modules can communicate, and client resource information can be retrieved from the client devices based on the interest list from different server applications.
- Client applications receive adapted data from the server applications as designed.

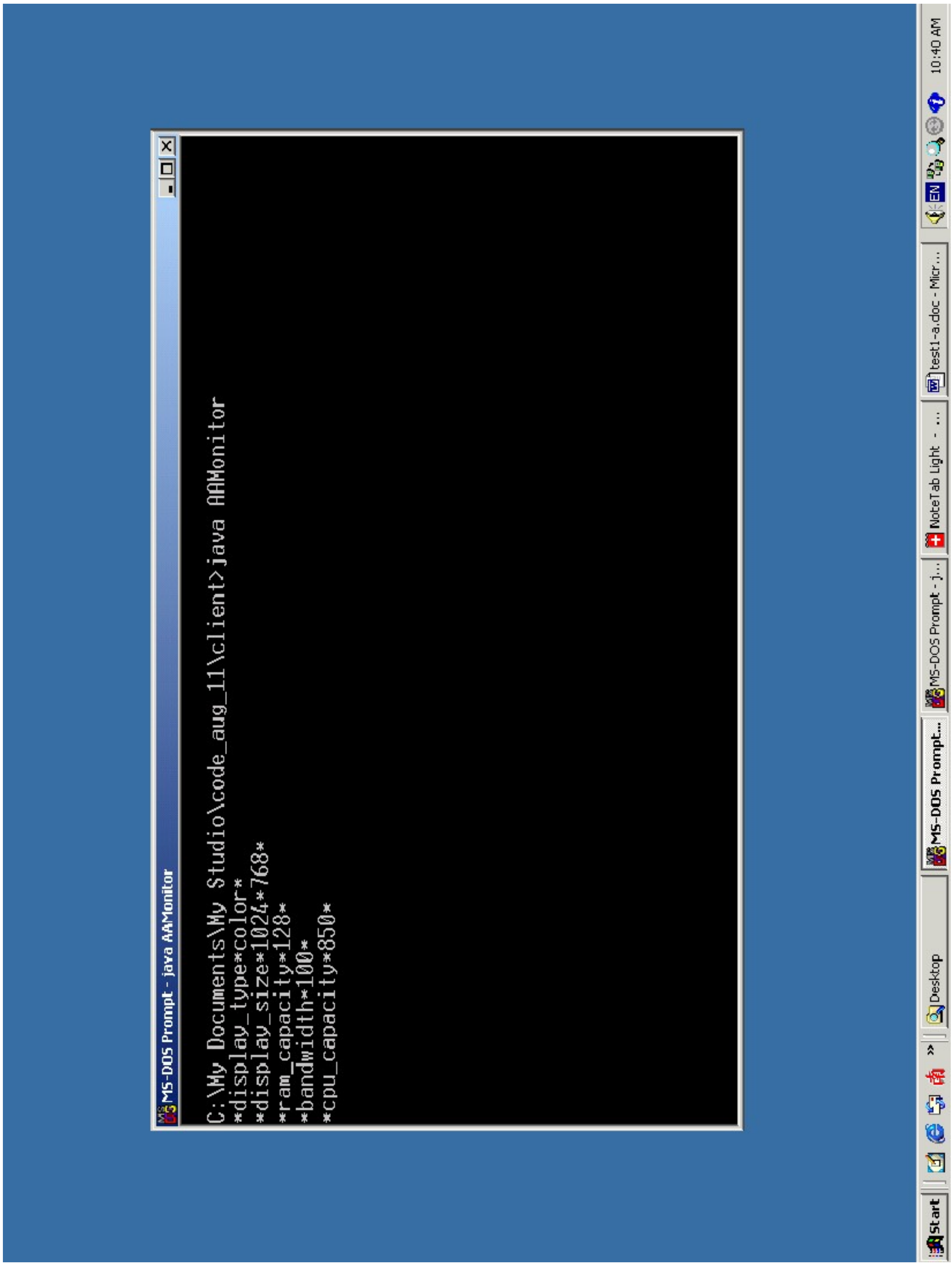

Figure 7.1: Test 1: Initial State of the AAMonitor

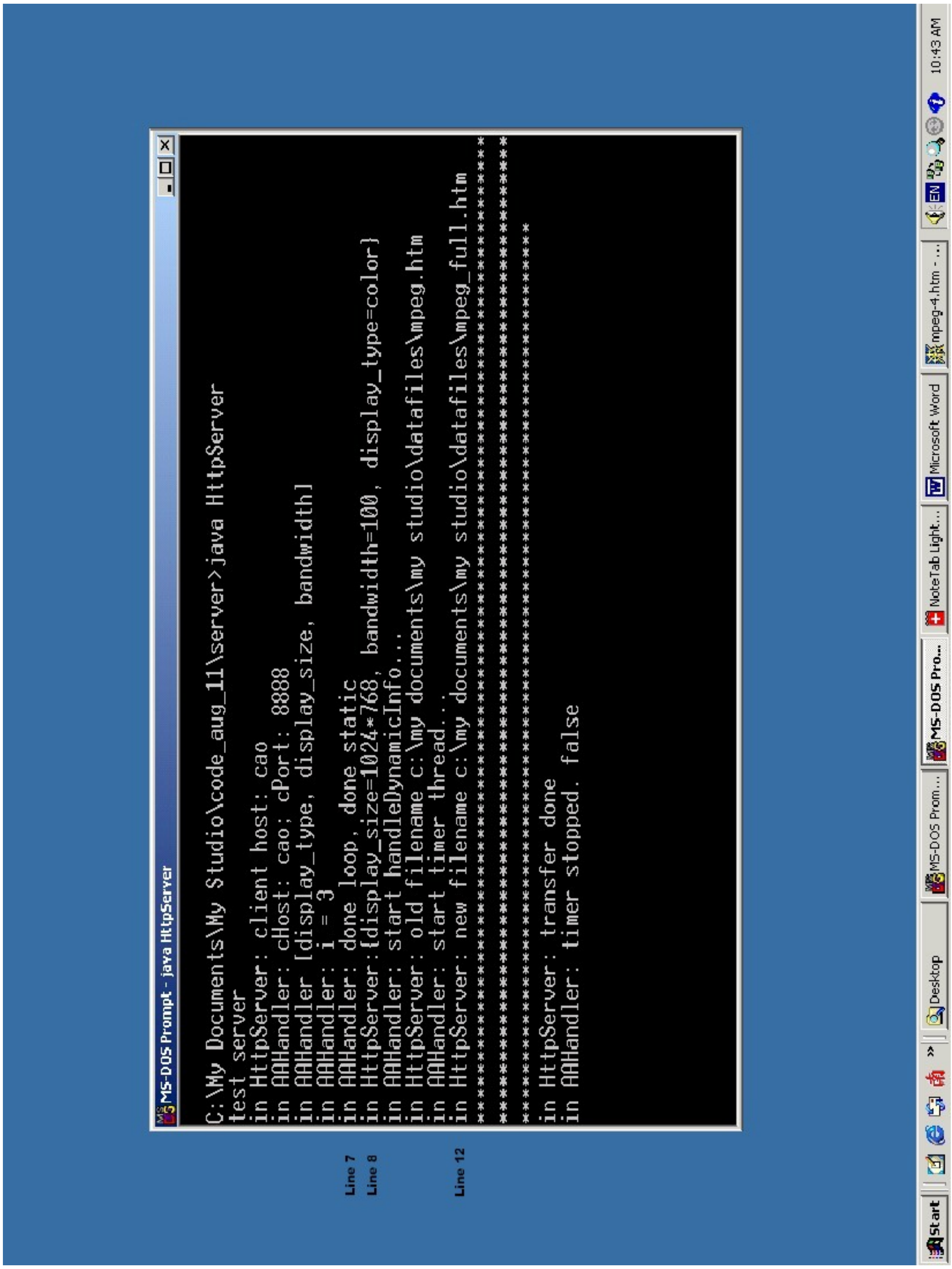

Figure 7.2: Test 1: Final State of the HttpServer

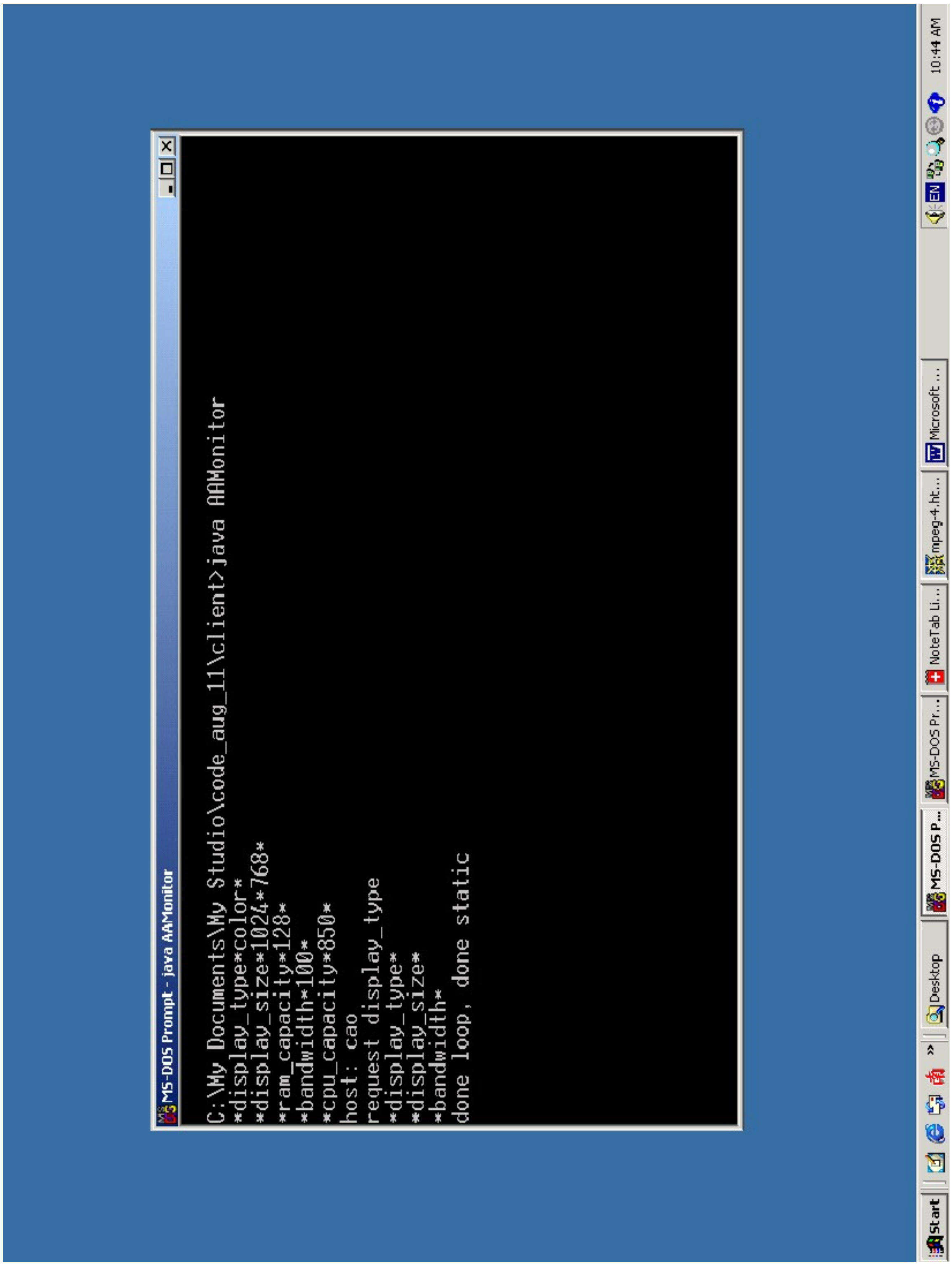

Figure 7.3: Test 1: Final State of the AAMonitor

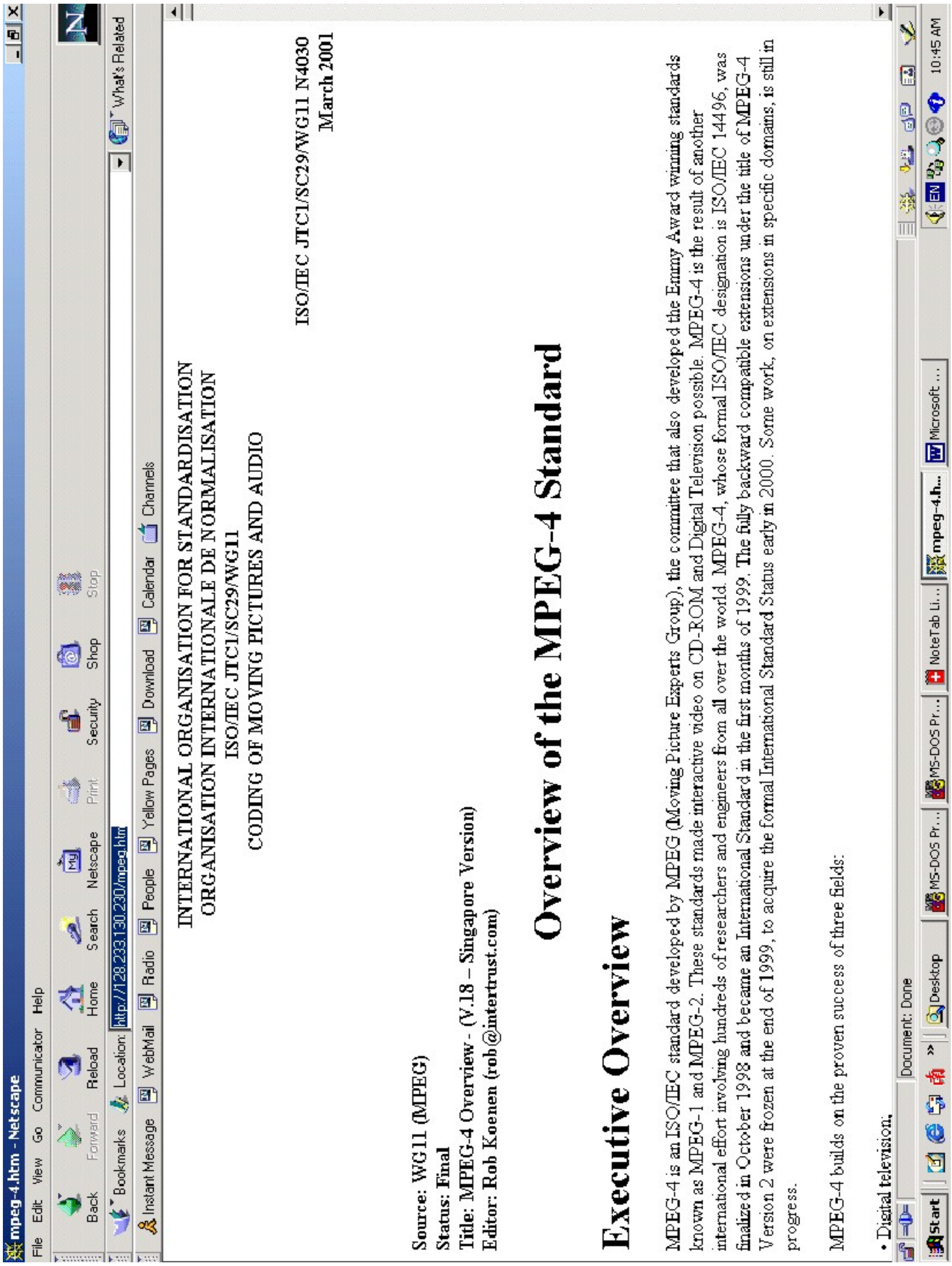

Figure 7.4: Test 1: Final State of the Browser

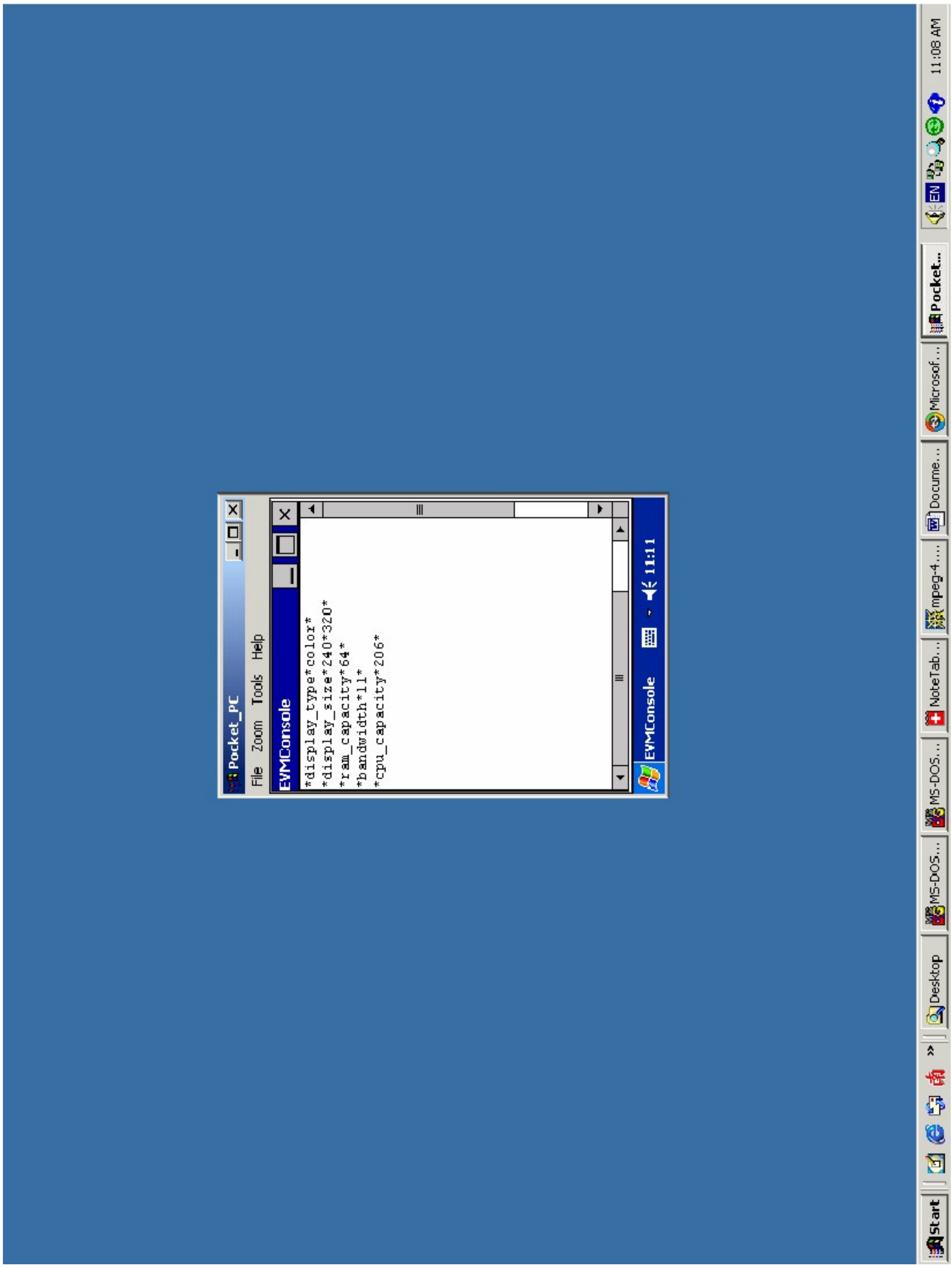

Figure 7.5: Test 2: Initial State of the AAMonitor

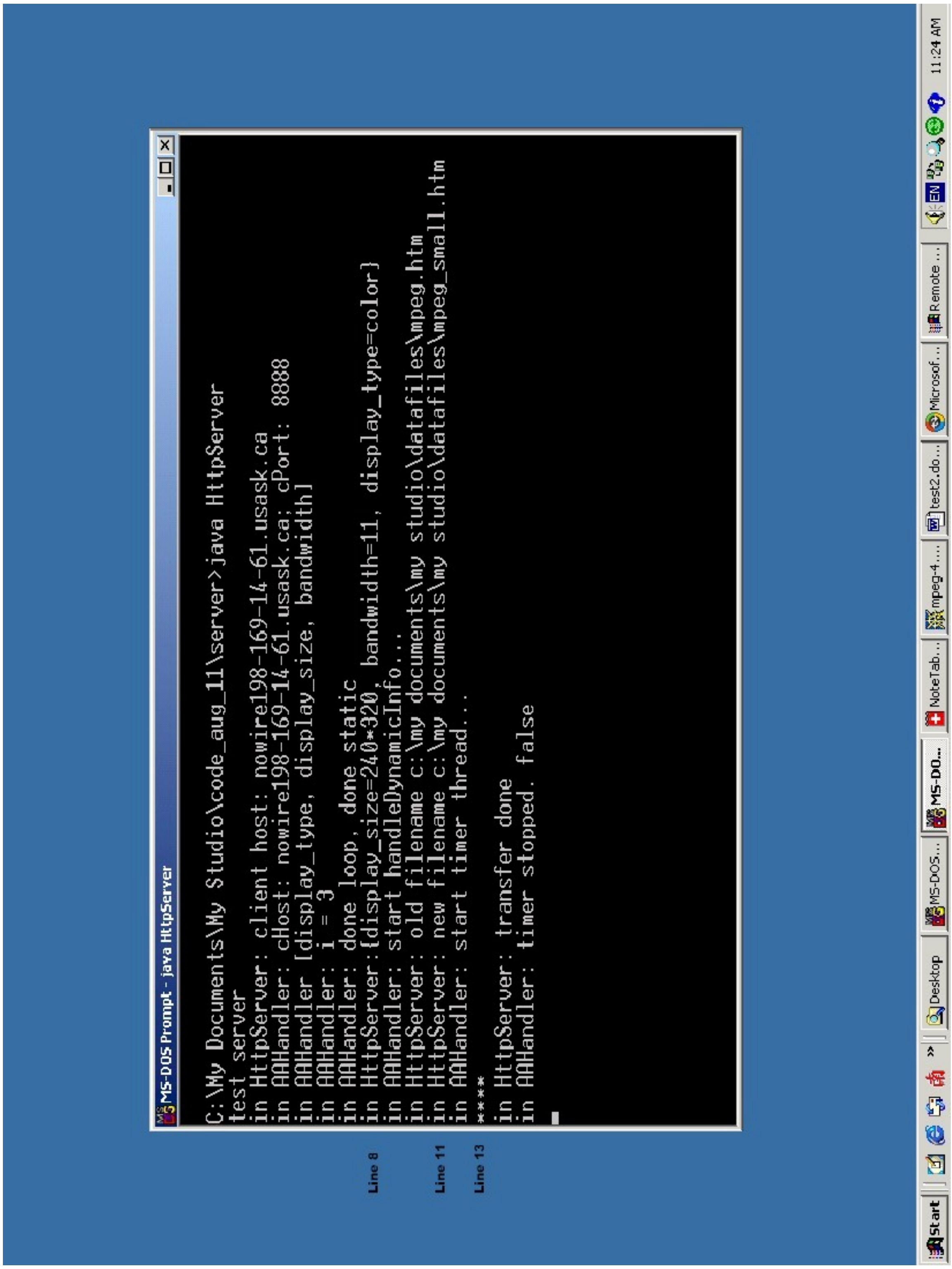

Figure 7.6: Test 2: Final State of the HttpServer

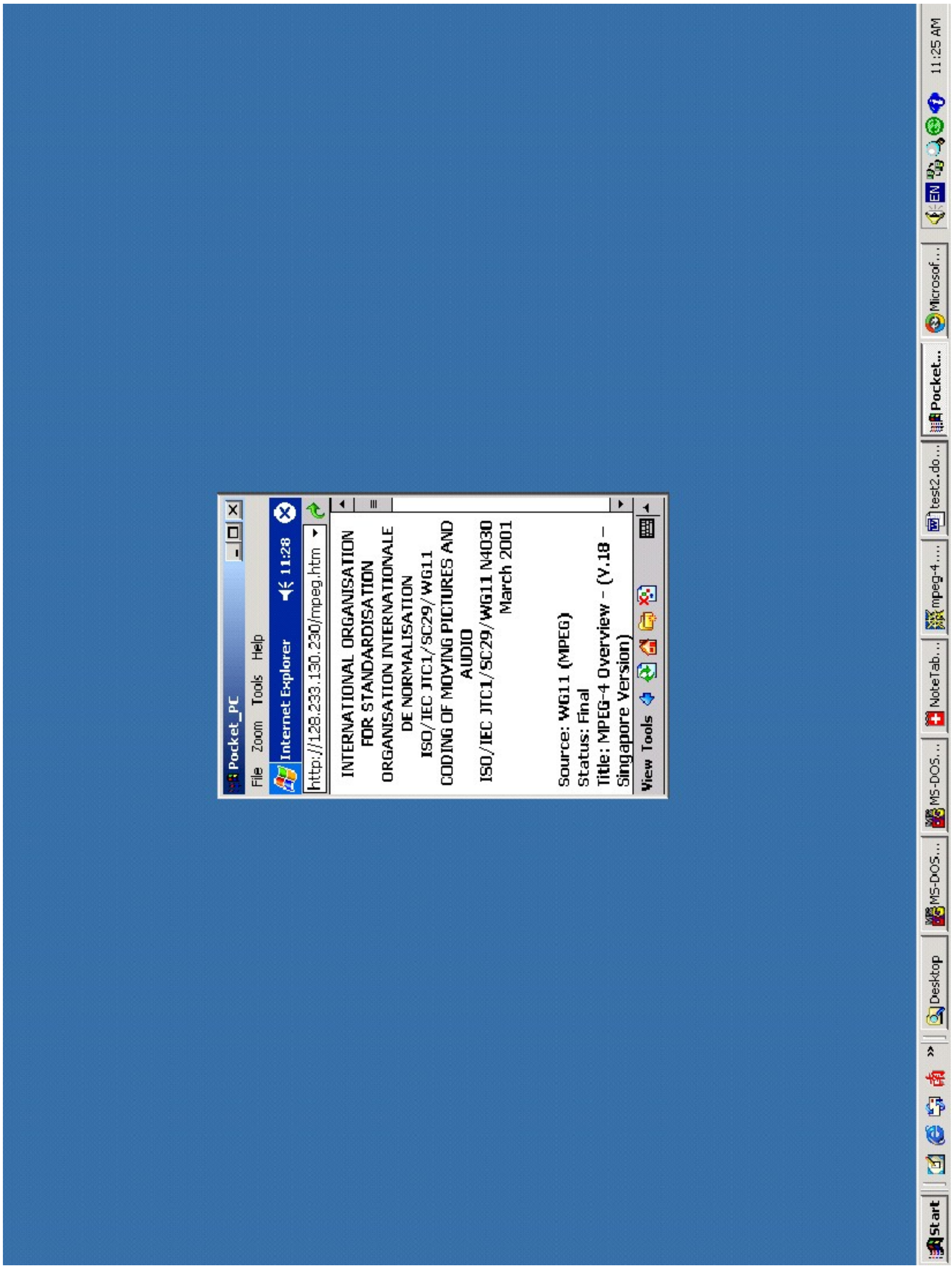

Figure 7.7: Test 2: Final State of the Browser

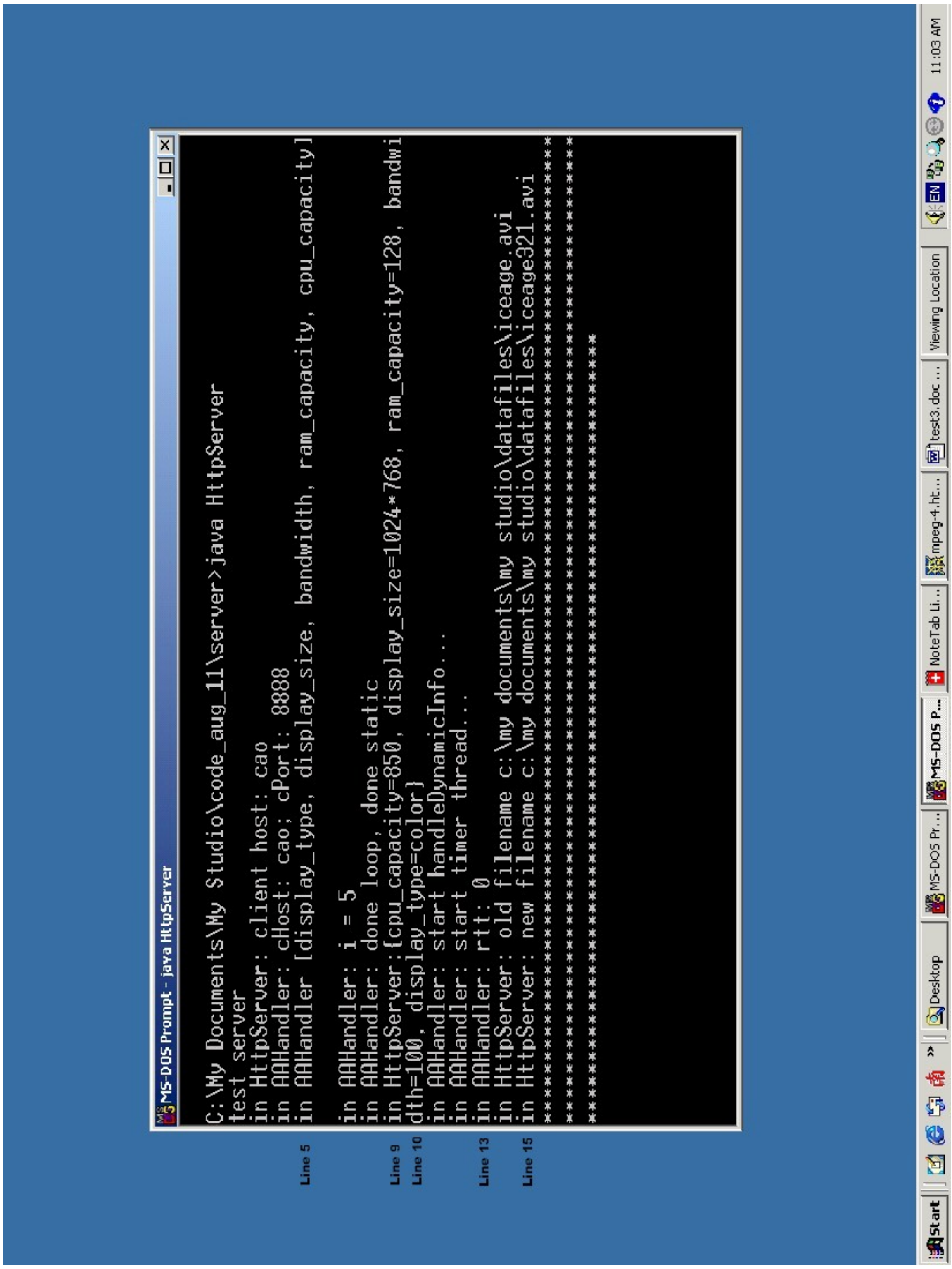

Figure 7.8: Test 3: Starting State of the HttpServer

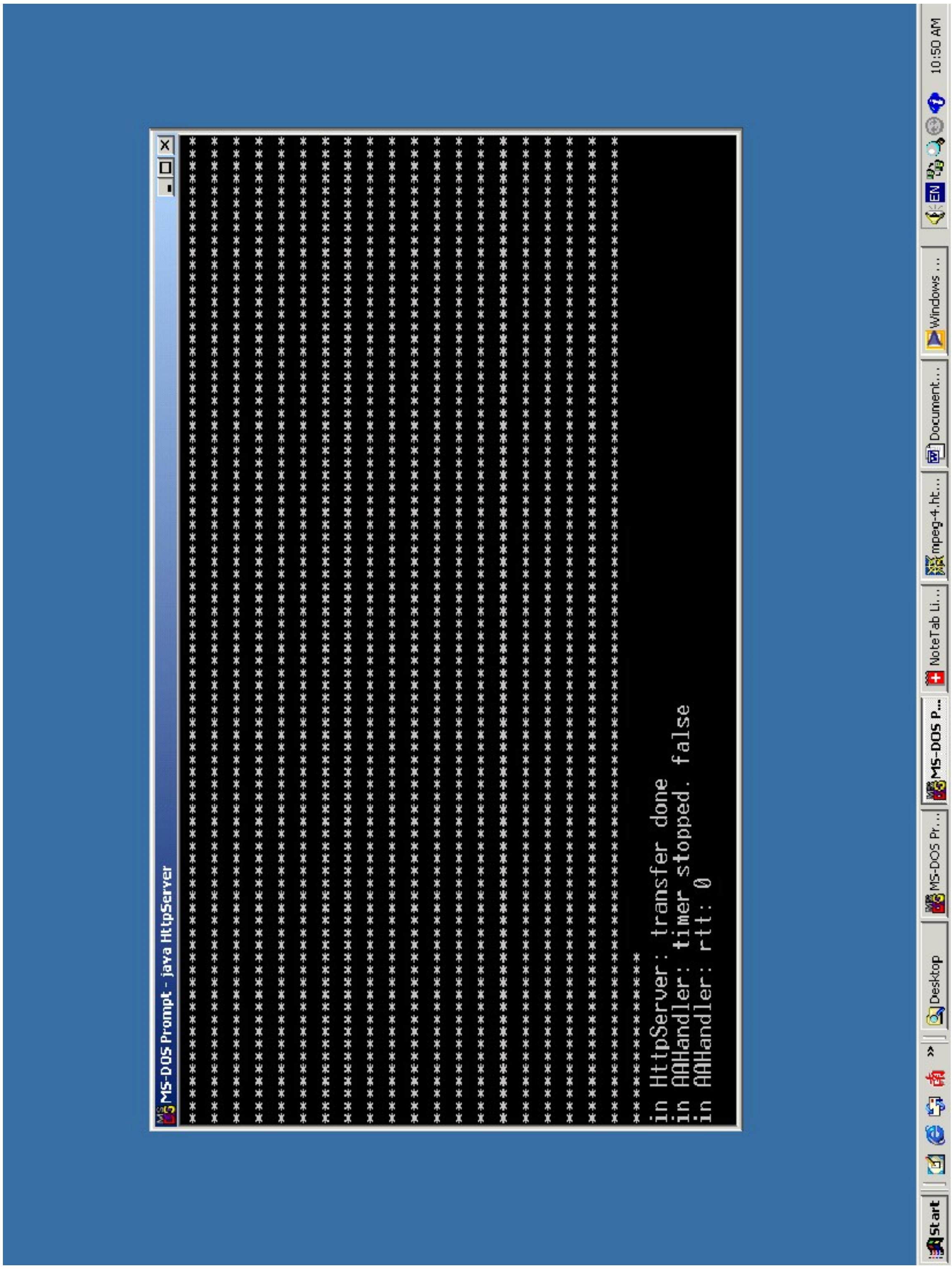

Figure 7.9: Test 3: Final State of the HttpServer

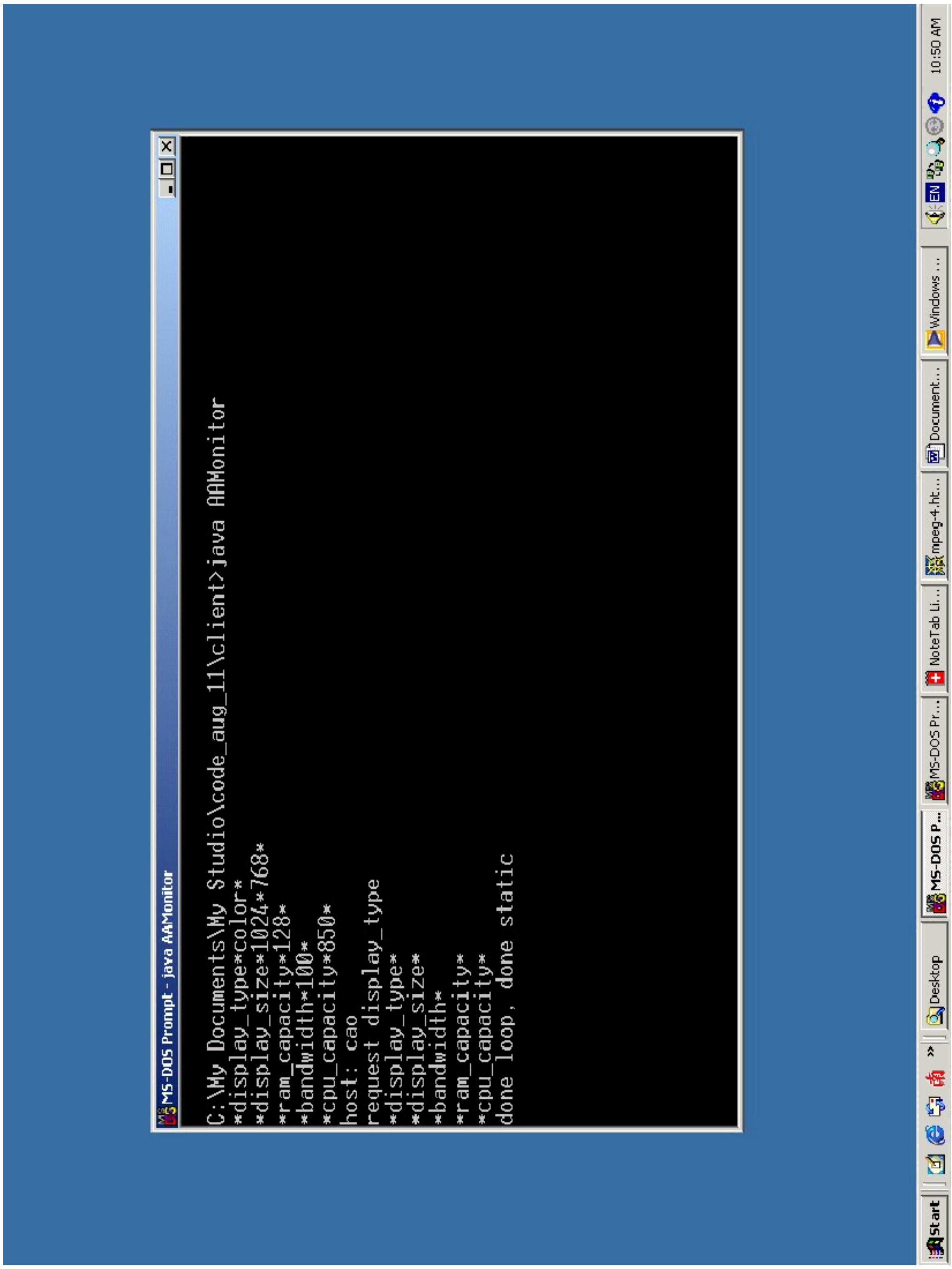

Figure 7.10: Test 3: Final State of the AAMonitor

Figure 7.11: Test 3: Final State of the Player – Playing the Movie

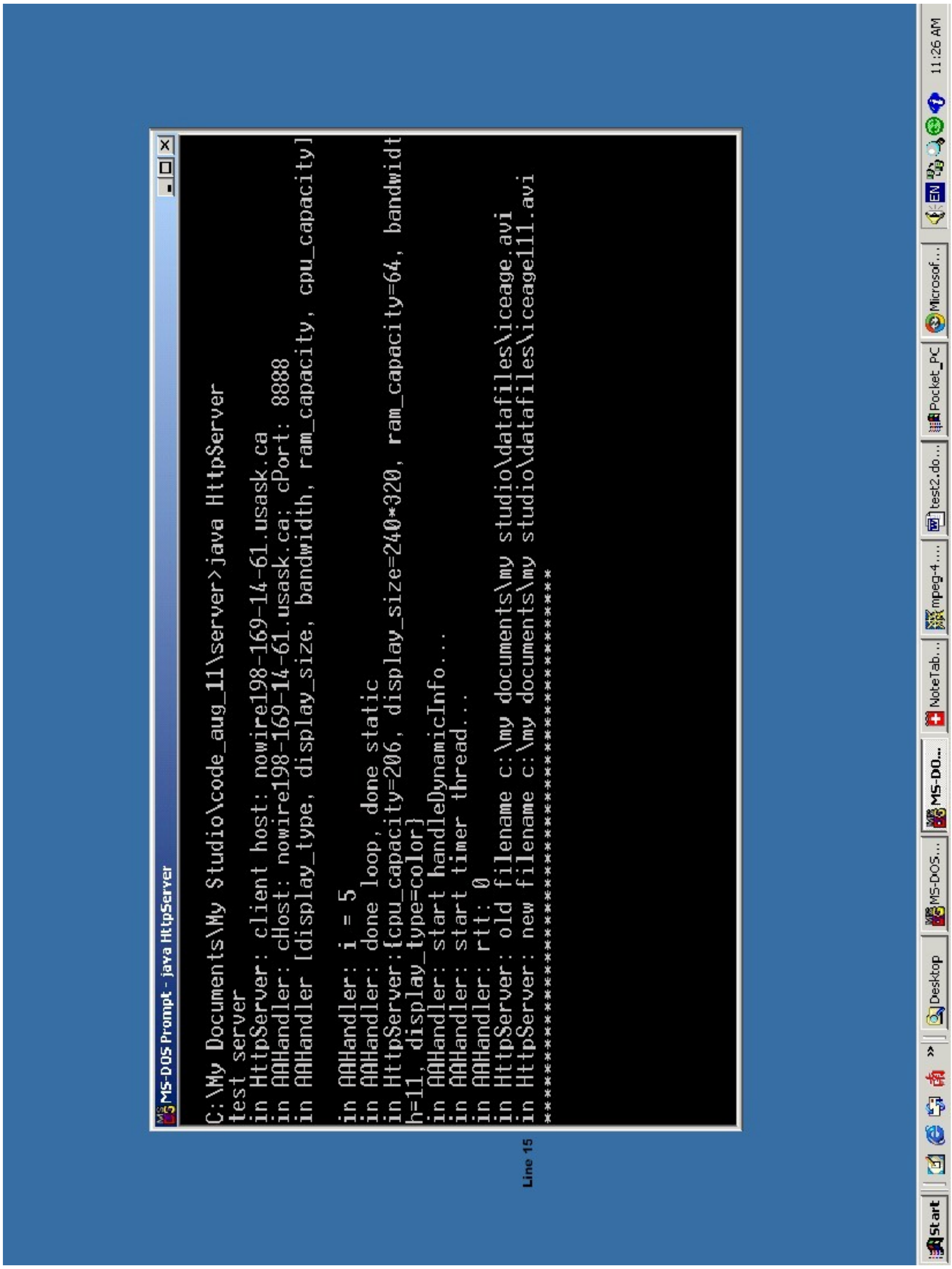

Figure 7.12: Test 4: Starting State of the HttpServer

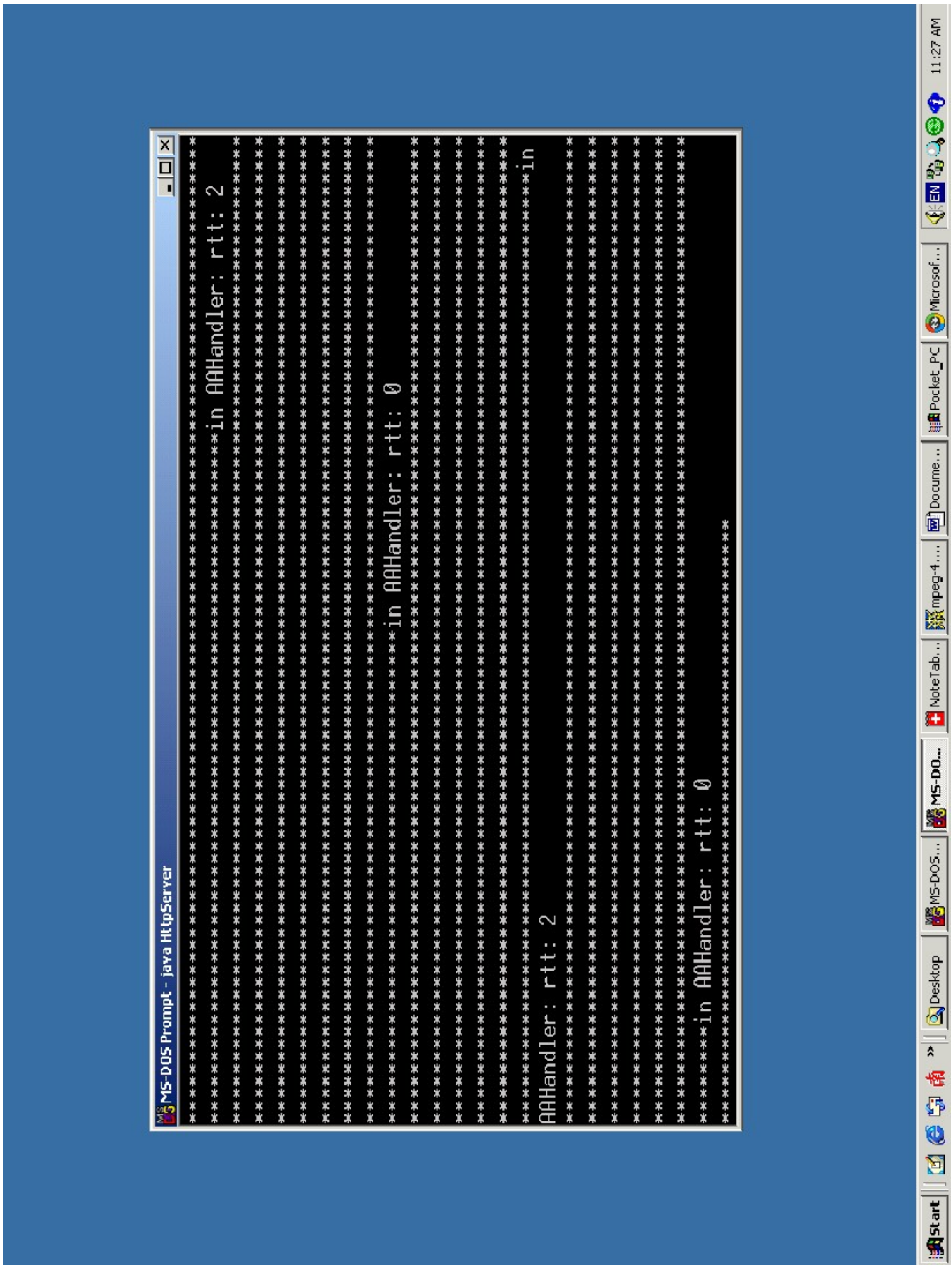

Figure 7.13: Test 4: Middle State of the HttpServer

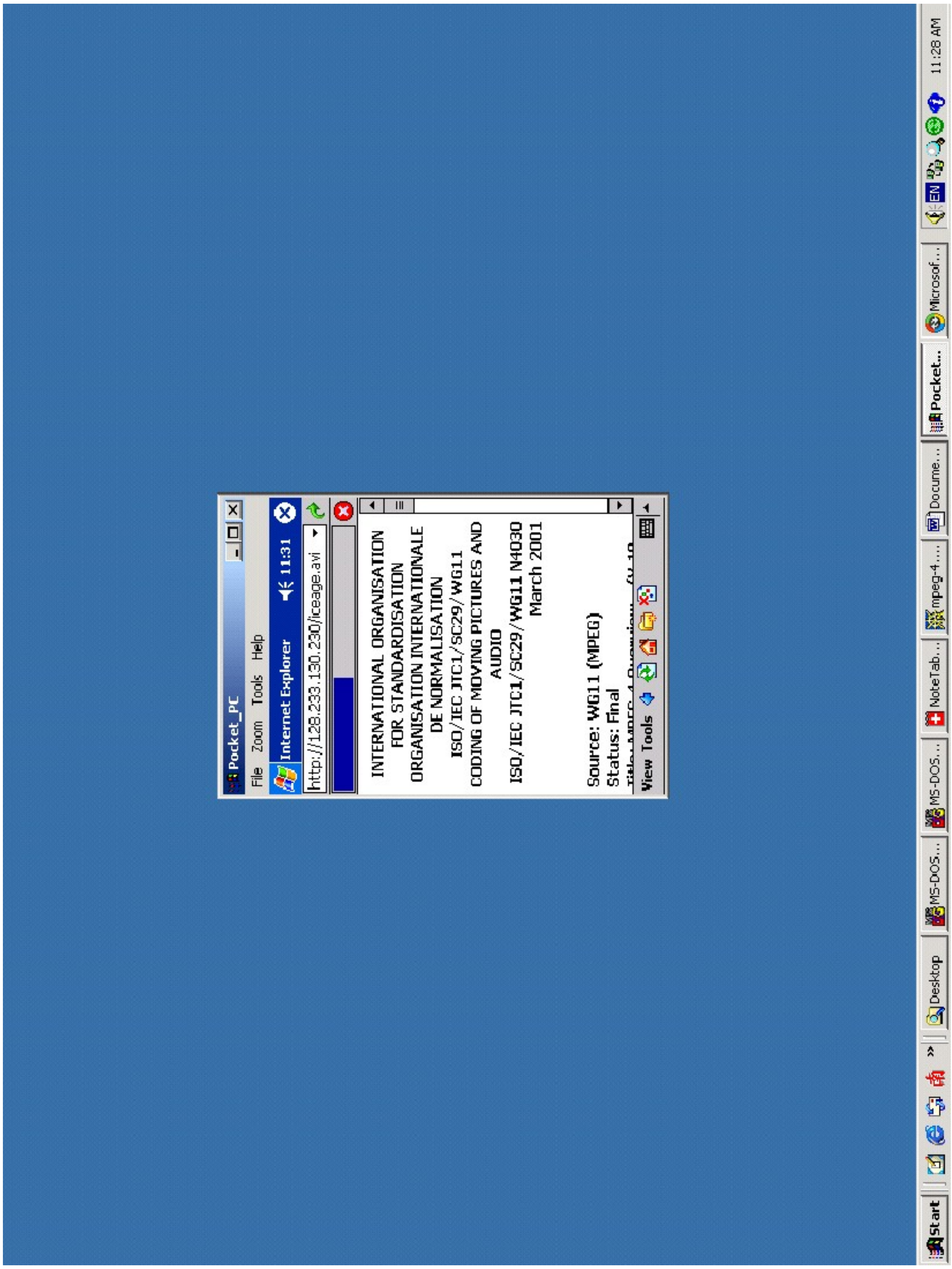

Figure 7.14: Test 4: Middle State of the Browser at Client

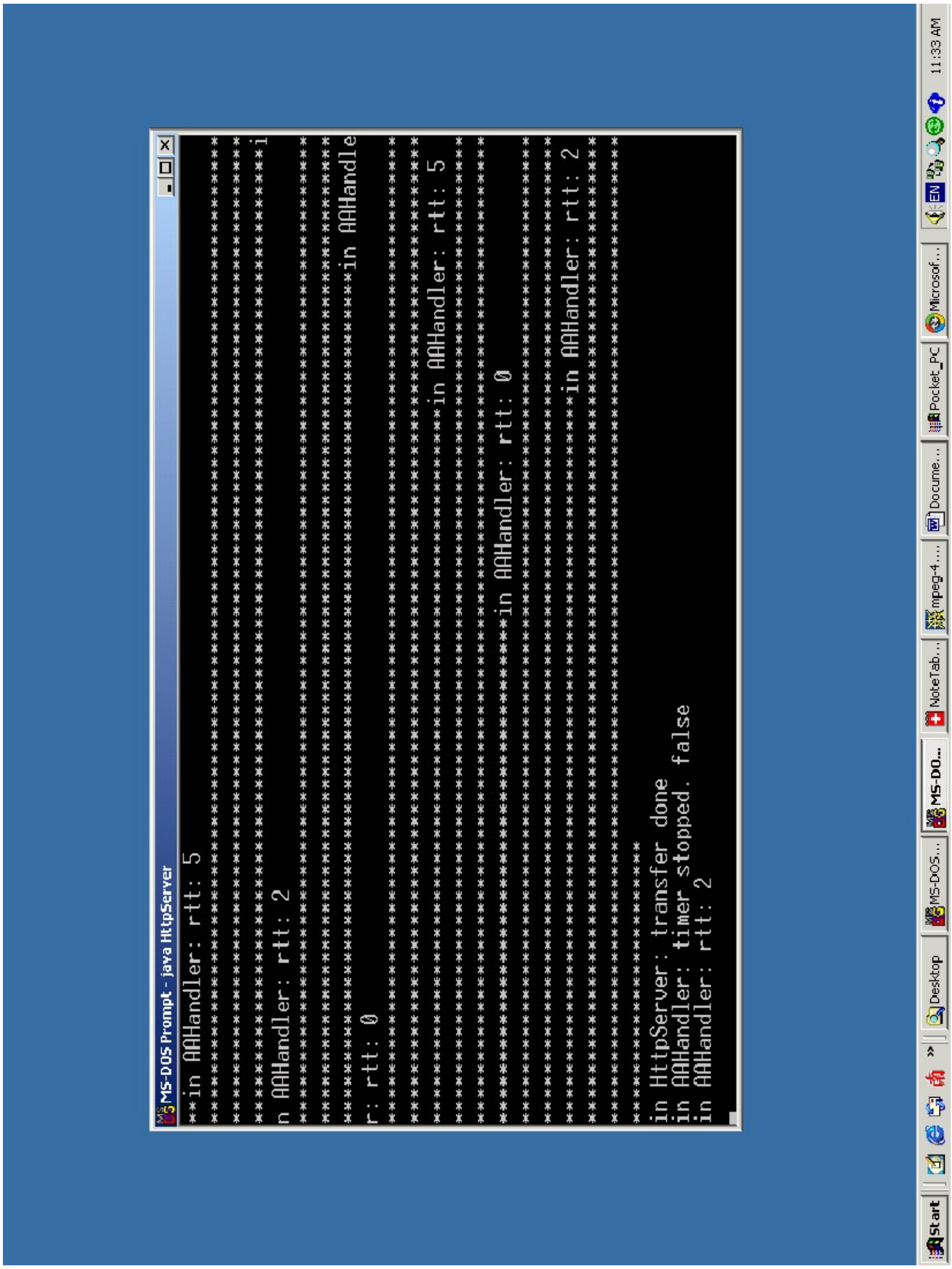

Figure 7.15: Test 4: Final State of the HttpServer

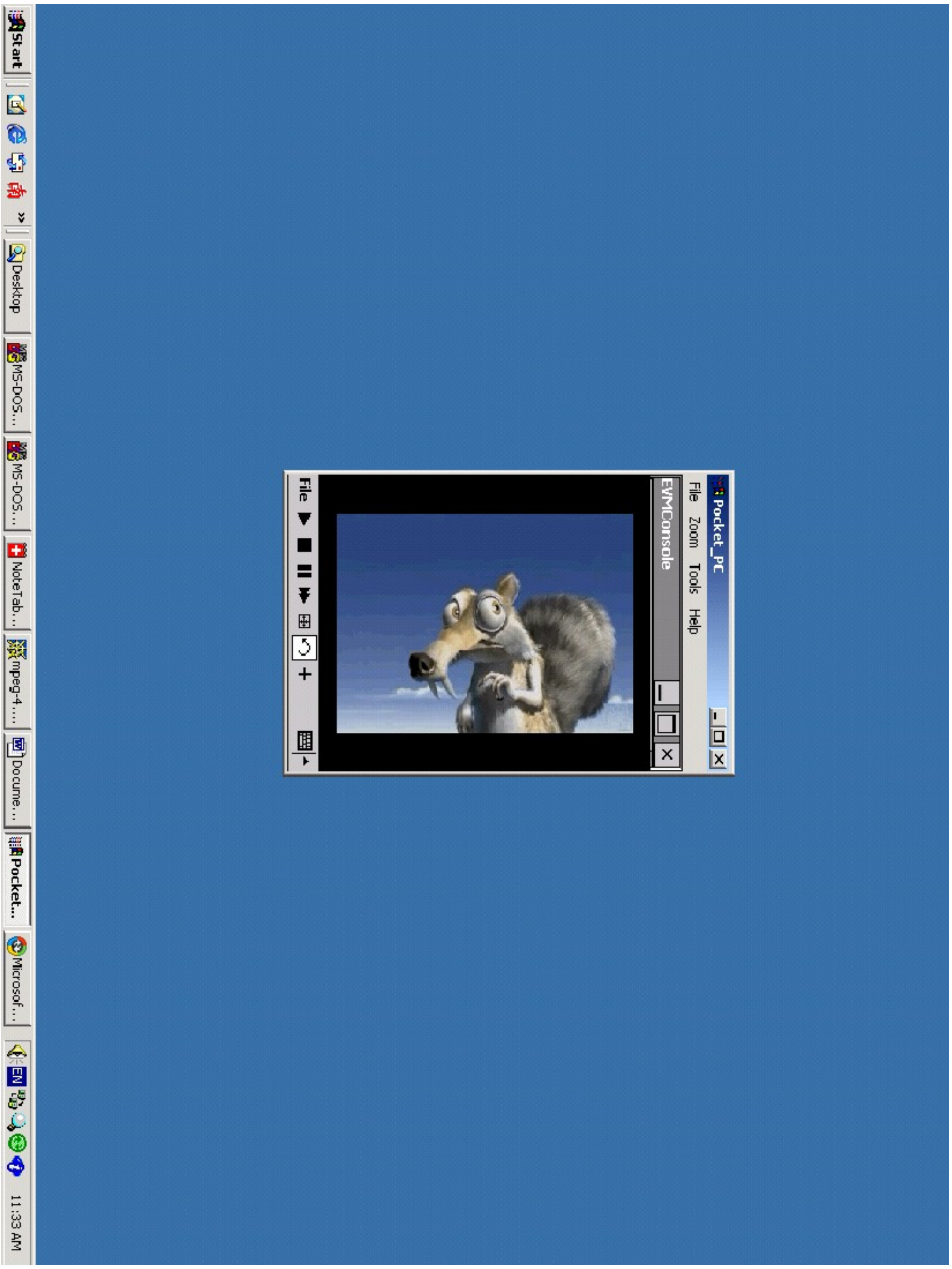

Figure 7.16: Test 4: Final State of the Player – Playing the Movie

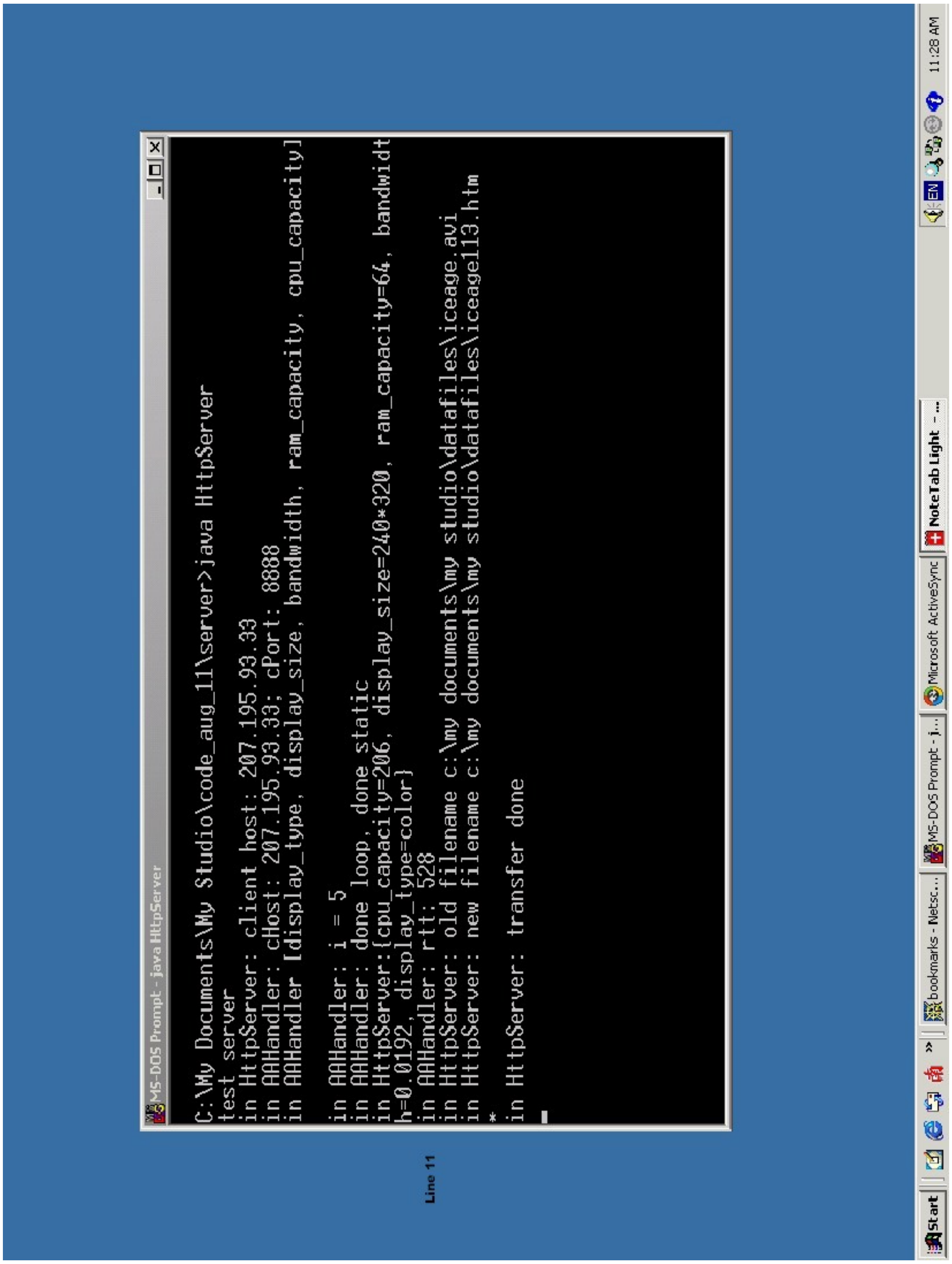

Figure 7.17: Test 5: Final State of the HttpServer

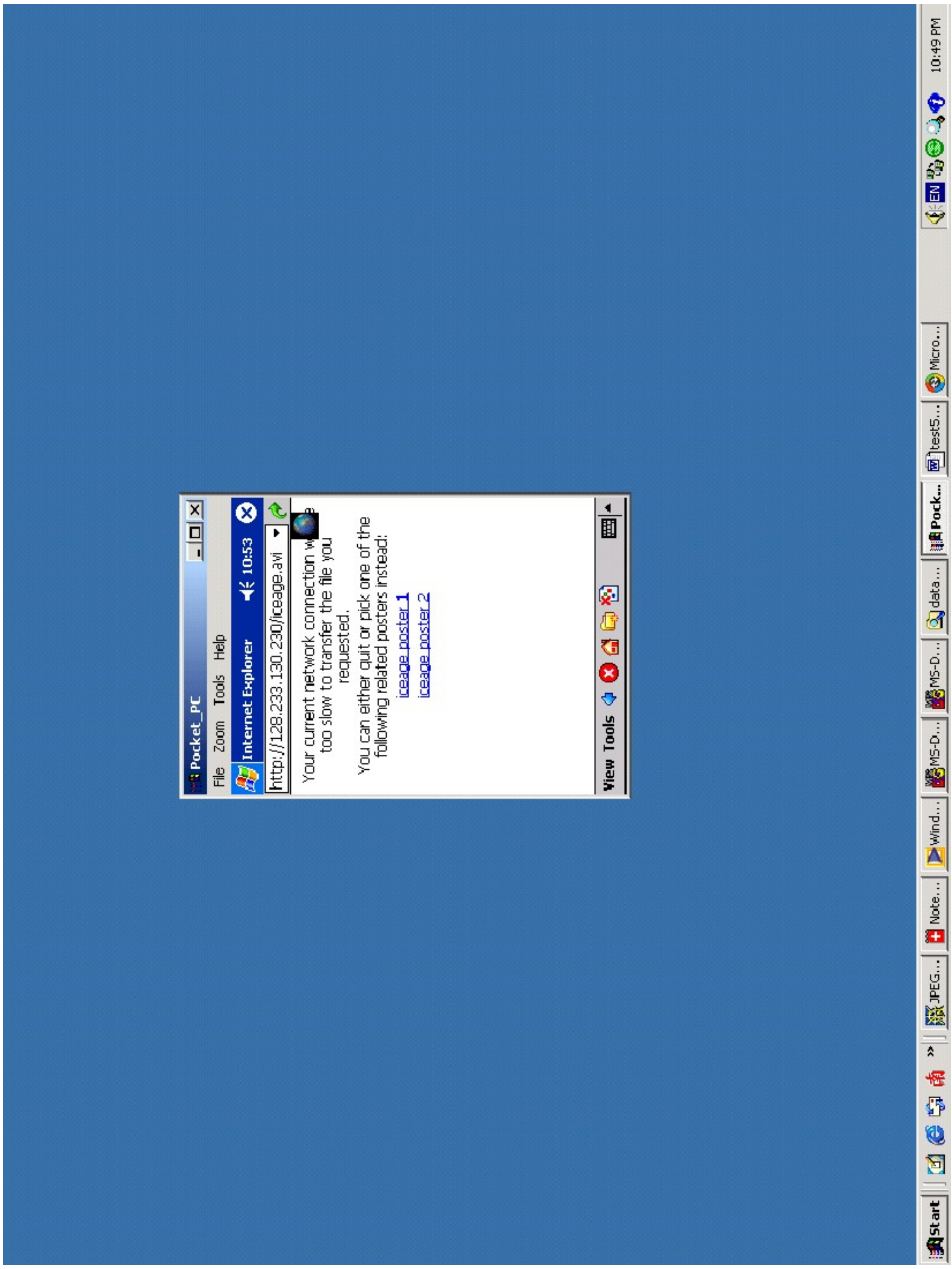

Figure 7.18: Test 5: Final State of the Browser – Showing the Information

Chapter 8

Analysis of the Design

The purpose of this research is to design an architecture that not only can be implemented but also is simple, effective, extensible, and cost effective. Evaluation of this design is done by analyzing its simplicity, extensibility, generality, cost effectiveness, transparency and performance implications. Comparisons are made to the Odyssey and TranSend systems.

8.1 Simplicity

Only one layer, the application layer, provides all necessary support for adaptation without modifications to lower layers (i.e., transport, network, or physical). Because the adaptation in the server application needs to consider only the middleware interface, this is a simple design to solve end-to-end performance problems. Solutions in lower layers could require upgrades of all the routers in the Internet, which is clearly not feasible.

The implementation is also simple in the sense that no modifications are required to the client application. Thus the complexity of the implementation is dramatically reduced. The implementation of the prototype middleware requires fewer than 500 lines of Java code. The upgrading of the server applications is straightforward. To attach to the middleware, fewer than 50 lines of Java code are added to each existing server application in this prototype. To implement the adaptation strategies, the major complexity is the application-specific component.

Generally speaking, it is hard to compare different systems in terms of simplicity since they target different goals and have different functionalities. Systems such as Odyssey and TranSend, however, are simple from the design point of view since they are also application layer solutions. But, since Odyssey is a client-based adaptation system, the cost of implementing such a system can be high since it might involve great effort to upgrade all client applications.

8.2 Extensibility

An extensible architecture can be modified to suit different environments. As is discussed in Chapter 9, Possible Extensions, this architecture can be extended either by adding new features or by moving one of the peer middleware modules to a proxy. Adding new modules to support features such as resource information management and resource information caching can improve the functionality of the current architecture. Also, with little effort in modifying the middleware and the server application, one of the peer middleware modules can be relocated to a proxy closer to the client.

These variations show the flexibility of this design. Special services can be supported as well, such as dealing with mobile devices that have no computing or storing abilities. Compared to Odyssey and TranSend, this design has better extensibility since Odyssey is a specific solution on client devices and TranSend is a specific solution on proxies.

8.3 Generality

Generality means that general data types, general applications, or general resource information monitoring can be supported within the design. The use of middleware provides general service for data types in that the adaptation strategies are managed by individual server applications. If an application-transparent approach is used, all the adaptation will be implemented by the operating system, and then only the data types considered prior to design time will be supported. Such a system may fail to recognize a new data type.

Interfaces provided by the middleware support general applications. New applications can be attached to the middleware by defining themselves as a subclass of the *AdaptiveServer* or by implementing the Java feature Interface *AdaptiveServer*, and by implementing the interfaces defined by the middleware.

The middleware solution provides general service for resource information monitoring. This research categorizes resource information into two types, static and dynamic, for convenient management and monitoring. New static information could simply be added onto the resource interest list and retrieved from the middleware on the client device, as long as the client device has the knowledge of the new type. When dealing with a new dynamic information type, both sides of the middleware must be modified or new modules must be plugged in. This upgrading procedure can be simplified by using the deployment centre that is described in Section 9.2.4.

Supporting general data types, general applications and general services for resource information monitoring is the key characteristic of an application-aware approach. Odyssey has good generality as a system using such an approach. In fact, it was the first system to apply the application-aware approach. Odyssey also provides a set of interfaces for general applications. New applications can be easily plugged into the Odyssey system. New data types can be added according to the requirements of an application [38]. TranSend is application-specific rather than general. Since the resource monitoring and adaptation strategies are implemented in the application on the proxy, only one specific application (i.e., a Web server) is supported [20] and other applications cannot take advantage of the resource monitoring.

8.4 Cost-effectiveness and Transparency

The design is highly cost-effective with respect to maintenance and implementation. From the administration point of view, server-based adaptation guarantees that the system will be set up and maintained centrally. Needing to maintain individual client devices can considerably increase the cost, due to the large number of clients and the unpredictable configurations of the client devices. Using a server-based approach reduces the implementation cost compared to the client-based approach, and using middleware to provide some general adaptation services also reduces the implementation cost by eliminating the need to repeat common code in every server application.

Transparency is not a firm criterion for a design. Some users would prefer that their input be taken into account, so as to increase the personalization of services, but for other users with poor knowledge of the computer or network settings, it could be rather annoying to have to manually set or reset the client device resource information prior to running an application.

Since those small mobile devices are designed to run mainly as a client, few users require detailed computer knowledge. Therefore, transparency is considered more suitable in this research. Although at the current stage, due to the limitations on the operating system support, transparency is not fully supported by the prototype, the architecture is designed to obtain the resource information from the client device by the middleware automatically without explicit input from the mobile users.

8.5 Performance Implications

Adding new functions to a server application introduces additional overhead at the server. This must, of course, be balanced by improved performance. Compared to other adaptation approaches, such as *ad hoc*, the middleware solution introduces little overhead. To convey both the static information and the dynamic information that applications need requires the transfer of quite modest amounts of additional data in comparison to the size of the content data. Also, the connection for static information needs to be set up only once, and the static information is transferred only once at the beginning of the connection.

For dynamic information, it is the middleware that takes the responsibility of monitoring and notifying the server application. The middleware will not be a bottleneck because it only passes client resource information, and so there is very little impact on server resources. Should there be a concern that many connections pass through the same middleware, these services could be provided using duplicate middleware modules instead.

Chapter 9

Possible Extensions

The current implementation has several limitations, partially due to the inherent constraints of the mobile devices and partially due to the design. The cause of these limitations and possible methods to solve them are discussed in this chapter.

9.1 Extensions to the Prototype

9.1.1 Current Technology Problems and Limitations

Problems encountered while implementing the prototype included the hardware limitations of the small mobile devices, the constraints on operating system support, and the software conflicts between the software support from the mobile device and the service from the server applications.

Various hardware constraints of the pocket PC were described in Chapter 2. As an example device used in this research, the Compaq iPaq has relatively poor computing resources of 206 MHz CPU and 64 MB RAM, a small display size of 240x320 pixels and relatively slow network connections when using wireless Ethernet and CDPD. These constraints make the iPaq not suitable for presentation of large images and contents involving substantial computing (e.g., decompressing) and buffering.

Ideally, the middleware on the mobile device can get the display information, memory utilization information and CPU utilization information without a user's awareness. Some static information can be obtained with Java, including operating system name, architecture and version using the java.lang.System class, but constraints of the Java design and Java implementation prevent the middleware from retrieving other important resource information automatically.

The display information, such as display width or display height, is not hard to get via the support of the JDK^{TM} 1.1 (Java Development Kit 1.1) or up on a resource-rich device.

The device-wide memory utilization information is impossible to get using Java currently since the java.lang.Runtime class supports only per-process and JVM^{TM} (Java Virtual Machine)-wide memory utilization information, including free memory and total memory of the JVM. This is a language-specific problem that cannot be solved by anybody other than Java Sun, since it is part of the design of Java.

The CPU utilization information is also not currently supported by Java. There are no utilities (i.e., programs pre-installed in the operating system that can be used by users from the command line) on Windows systems to provide CPU utilization or memory utilization. It is always easier to get numeric information using utilities such as top and stty on Unix/Linux systems, but these are not available on pocket PCs at the present time.

Software limitations make it difficult to implement a real-time video server which can take advantage of the dynamic information such as transfer rate or RTT since each frame of the video clip must be controlled on the fly.

One possible way to implement the video server is to use the JMF package [47]. JMF is a large and versatile API used to process time-based media. Implementing RTP (Real-Time Protocol) and RTCP (Real-Time Control Protocol) using JMF requires only a few lines of code. The problem with using JMF is that it provides only high level control of the video streams. In this research, some of the adaptation strategies need to access individual frames, which means a decoder is needed to get an uncompressed frame and an encoder is needed to get a compressed frame after the frame has been modified and before being sent out to the network. JMF itself does not contain an MPEG decoder, even though it can import them easily. There are MPEG decoders available, either in C, C++ or in Java, but no MPEG encoder implemented in Java is available currently.

Another problem is that the server written in JMF could not communicate with the media player used on the client devices. In the case of an iPaq as a client device, JMF is not supported by the current Java platform on iPaq. The only media players currently running on iPaq that support video streaming are PocketTV, DivX Player and Windows Media Player v7.1. The RTSP (Real-Time Streaming Protocol) supported by Windows Media Player does not match the RTSP provided by the JMF server.

Video compression method also affects the difficulty of the implementation. Three current commonly used compression methods are MPEG, JPEG and H.263. MPEG was started in 1988 as a working group within ISO/IEC with the aim of defining standards for digital compression of audio-visual signals. Among the MPEG family, MPEG-4 is designed specifically for digital television, interactive graphics applications and interactive multimedia [37], and therefore is preferred currently. It is hard to find a free MPEG-4 codec since it is still a new technique. Some claimed to be MPEG-4 but were actually not.

The H.263 video compression standard, published by the ITU (International Telecommunications Union), supports video compression (coding) for video conferencing and video telephony applications [29]. JMF supports this compression format, but there is no player on iPaq that can play this format yet.

Since JPEG defines only still image compression, it is not considered a good example of real-time video format, and will not be used in the next step of this research.

9.1.2 Solvable Problems and Possible Extensions

Some of the problems with the current stage of the prototype are solvable, some of the constraints can be bypassed, but some are very hard to avoid. The following extensions can be done in the future.

Microsoft eMbedded toolkit can help developers writing applications for pocket

PCs in C++. The current version of the Windows System Information package supported by the Microsoft SDK (Software Development Kits) provides operating system interfaces for retrieving hardware information (e.g., processor type, display width and display height) and operating system information (e.g., operating system version). But the question remains whether all of those interfaces are also supported by Windows CE. If so, retrieving operating system information automatically by middleware can be implemented in a C++ version of the middleware component for pocket PCs.

To add the feature of adapting to dynamic information in this prototype, a real-time video server can be a good example application. JMF is a reasonable next choice to build a real-time streaming video server because of its standardization. Even if there is no player supporting this real-time streaming on pocket PCs currently, this server at least could communicate with a player on other mobile devices, such as laptops, that have more resources.

As an example application, not all of the dynamic information has to be taken into account, but at least the RTT, memory load and CPU load should be considered by the adaptation strategies used by this adaptive server application.

This prototype is based on Windows systems. Since some server applications are based on Unix/Linux systems, to provide services for these applications the middleware on the server side should work in the same manner on a Unix/Linux platform. Some minor modifications would be required to the current version of the prototype. For example, the path of the configuration file should be in a different format on a Unix/Linux system. But since Java is also supported by Unix/Linux, there will not be significant changes.

9.2 Extensions to the Architecture

This architecture can be extended in a number of different directions: either by varying the place that the middleware sits or by adding new features. With respect to the place, the options are to move the server middleware to a proxy closer to the client or to move the client middleware to a proxy closer to the client, acting as a client agent. For new services, resource management and deployment centre are introduced as examples of new features. These variations are described in this section.

9.2.1 Moving Server Middleware to a Proxy

The server-based adaptation design can be modified to use a proxy close to the client devices. The most straightforward approach is to move the middleware to the proxy and replace the "server application" with a program that acts as a client to the server and as the server to a client. The middleware module on the proxy can still communicate with the middleware module on client devices and provide services to the new program similar to those it provides to a server application on a server machine. In this variation, the new program on the proxy provides adaptation services to the client devices based on the resource information provided by the middleware.

In such a proxy-based variation, the adaptation will be applied based on the resource information from the client devices, including the static settings and the dynamic information such as CPU load and memory load. The network conditions will be estimated based on the connection between the proxy and the client device, which is usually a weak wireless link. Since the original data has already been delivered to the proxy and has been buffered by the proxy if necessary, it is reasonable to send the adapted data from the proxy to the client based on the network conditions of this part of link only.

By using the middleware, general applications can be supported by this proxy-based adaptation solution. All the other advantages of proxy-based adaptation mentioned in Chapter 4 are also inherited. The most important advantage is compatibility with the existing server and client applications.

9.2.2 Moving Client Middleware to a Proxy

The main limitation of the architecture design is that it cannot support mobile devices which do not have computing ability such as Sun Ray^{TM}, a thin client device produced by Sun. This criterion cannot be supported by either the server-based or the client-based system designs. There is an easy way to solve this problem by making the middleware a client agent.

The middleware module which is originally installed on mobile devices cannot run on those mobile devices with no or very limited computing/storing capacities, such as a smart phone. However, by migrating this middleware module to a proxy closer to the mobile devices, the architecture would still work well.

In this variation of the architecture, the proxy should have knowledge of the resources of the mobile devices, which can be obtained by the middleware. The middleware acts in the same manner as when dealing with the static information. For those server applications that require dynamic CPU and memory information, values to represent the poorest resource level will always be sent. For those server applications that require a consideration of the network transfer rate, since the network transfer rate will be estimated by the connection between the peer middleware modules, the data will be delivered to the proxy based on the network conditions between the server and the proxy. This delivery does not take into account the connection between the proxy and the client device, usually a slow wireless link. This problem can be mitigated by buffering the data at the proxy if necessary.

9.2.3 Resource Management

If more resources are taken into account, a resource information management component will be needed, as part of the middleware, to provide better services. Resource information management also involves exploring the relationship between resources and adaptation strategies. Some tentative proposals include sophisticated management of the resource types, providing interfaces to add new mechanisms for new dynamic resource types and caching on the resource information.

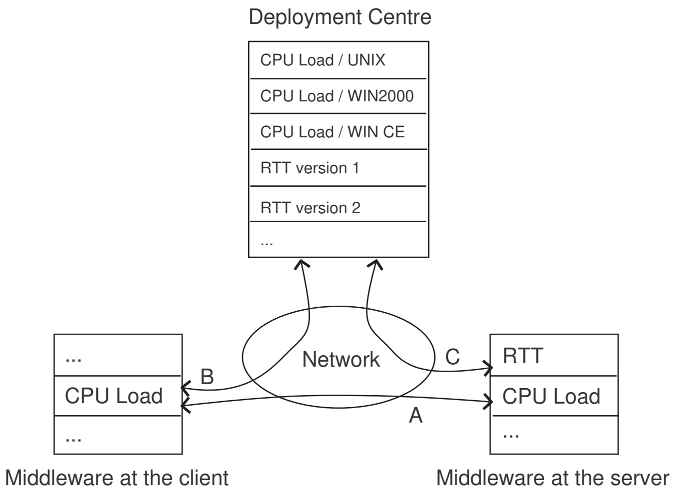

Figure 9.1: Deployment Centre

Resource information caching is an attractive technology since when the same client device connects the server multiple times (in the case of visiting a html page containing images and videos other than text, multi-connections will be processed when using HTTP 1.0), the bandwidth could be saved by reducing the repeating transferring of static information from the same client device, and the response time will also be reduced.

9.2.4 System Upgrade Using a Deployment Centre

Currently, components to support new resource information types and new resource monitoring/retrieving schemes cannot be added dynamically when the mobile applications and the middleware are running. A manual upgrade of the middleware and reboot of the system are required. This problem can be solved by using a control process acting as a deployment centre, which can sit in the server machine or at a different place in the network.

Figure 9.1 shows the relationship among the deployment centre, the server and the client. When the middleware notices the requirement for a new component of a monitoring/retrieving scheme or a new version of a component, the middleware can automatically download the required component from the deployment centre. There are two common cases that could take advantage of the deployment centre.

The first case is when the middleware needs a new component to support a new resource type. According to a request on a new resource type (see line A in Figure 9.1), for example, the CPU load, the retrieving schemes are different due to the variety of the operating system of the mobile device. The middleware at the client needs not only a new component to support this new type but also the proper version that can run on the current operating system. A deployment centre can provide such service for the middleware to download the proper new component dynamically (see line B in Figure 9.1).

The second case is when the middleware needs a new version of an existing component. For example, the current prototype uses "ping" to get the average RTT. When there is a better way of getting the RTT, a newer version, RTT version 2, can be maintained by the deployment centre for the middleware at the server to download dynamically (see line C in Figure 9.1).

To communicate with the deployment centre, some new protocols are required. One crucial protocol is to check if there is a required component available or if there is a newer version available. Another protocol is needed to determine which protocol (i.e., FTP or HTTP) to download.

There are also design issues on who initiates the request on checking a new version and how often to check. The request can be initiated by the middleware when needed or by the deployment centre whenever a new version is available, called *on-demand* and *broadcast*, respectively. The broadcast approach is ideal since it can guarantee that the middleware always has the newest version. However, this approach is not realistic in implementation due to the difficulty in maintaining all the middleware modules in the network. The on-demand approach is more realistic to be implemented, but how often to check remains a question.

Chapter 10

Summary, Conclusions and Future Work

10.1 Summary

The goal of this research was to develop an architecture that facilitates the adaptation of server applications when dealing with heterogeneous mobile devices and experiencing changing resource conditions. Four major design principles were developed to realize this goal.

The first principle is to introduce adaptation services into the application layer of the traditional network model. Since the applications are the end points of the connection, this is a convenient place to provide adaptation services.

The second principle is that both the variety of communication abilities and the heterogeneity of the mobile devices themselves are to be taken into account. The requirements of the mobile applications also vary and this architecture can incorporate such diversity.

The third principle is that adaptation be server-based. Compared to client-based adaptation, providing adaptation by server applications can reduce the complexity of modifying different client applications. Resource-poor devices that cannot support client-based implementations can be supported by this design. Server-based adaptation can also support multiple server applications by defining a set of standard interfaces.

The fourth principle is that services be provided through middleware provides resource information services for adaptive server applications in a clean and efficient way. The performance of a server application can be improved by sending adapted data based on resource information. The architecture in this research applies a middleware module and divides the responsibility of adaptation between the operating system and the server application. This approach reduces the complexity of modifying individual applications and increases the central control of the operating system by balancing between *ad hoc* approaches and *application-transparent* approaches. Using middleware also enables flexible extensions, such as moving the middleware from the server to a proxy or from the client to a proxy, to achieve similar adaptation support in different scenarios.

10.2 Lessons Learned

Some important lessons have been learned while designing the architecture and building the prototype.

- Seven design issues are discussed in this thesis on how to design a proper system in different network scenarios. Because of the wide variety of mobile applications, the answer to these questions is always "it depends." There are advantages and disadvantages to any possible solution. When there is a spectrum of possibilities between two solutions, certain application-specific conditions may influence the developer to choose an approach closer to one end of the spectrum or the other. These considerations have been addressed in Chapter 4 and Chapter 5.

- Simple does not mean bad. This design is quite a simple solution to support adaptive mobile applications. Heterogeneous mobile devices, including one of the popular pocket PCs, Compaq iPaq, can be supported under changing network conditions. The prototype and the analysis show the feasibility of this approach and the extensibility of this design.

- It is easier to come up with a design idea than it is to implement a prototype. The difficulties come mainly from mismatching of data types or the protocols between applications and from lack of support from the operating system. These difficulties represent common problems in the current technology for mobile computing. Some of the difficulties can be solved by improving software, whereas some can be solved only by improving technology.

10.3 Future Work

Future work includes extending the architecture and improving the prototype. Possible extensions on the architecture design were described in Chapter 9. To improve the prototype, automatic resource information retrieving on the client and dynamic adaptation offer considerable promise.

This work has also opened four topics as future research directions. How can the performance of the design be improved? How can adaptation strategies best use the resource information? How can streaming video applications adapt dynamically to changing resources? How much can the resource-aware adaptation overlap user-aware adaptation and context-aware adaptation?

Solutions to improving the performance of the design can be found in efficient implementing (reducing the overhead), improving monitoring schemes, or cooperating with lower layers of the network. Efficient implementing includes using a more efficient language, such as C or C++, and reducing the overhead of the middleware at the client device. Using a language other than Java could require re-design of some parts of the architecture. For example, because C is not an object-oriented language, the interfaces are different, and the mechanism to notify the dynamic information is different as well. However, the protocol to retrieve the static information remains the same. How to implement efficiently and how much performance can be improved are still questions.

Improving dynamic resource monitoring schemes can better the non-functional performance to the mobile users. Since the dynamic resource monitoring spans the whole transferring time frame, the more frequent the sample rate is, the more bandwidth and computing resource are spent on monitoring. For example, the current scheme to monitor RTT is using "ping", which involves initiating a new process, redirecting the output of "ping" and extracting the useful information from the redirected output. A better scheme is to time the packets sent and received when retrieving the static information or when delivering the content data, and then, to calculate the average as the RTT.

In some cases, it is also necessary to cooperate with lower layers of the network to improve the performance. For example, the alternative scheme to monitor RTT requires getting the information on packets from the transport layer. Another case is in the use of a wireless network, in which the error rate is traditionally higher than that of a wired network. The error control of the TCP/IP suite that widely used in current wired Internet means retransmission of failed packets. In the case of streaming video data, retransmission is not necessary all the time and even harmful when it causes the delay of later frames of the video data. Since delayed packets will be dropped at the client anyway, retransmitting could be eliminated sometimes to save bandwidth. However, how to balance the dropping of video data at the server in the application layer, at the routers in the network layer or at the client in the presentation layer, needs to be further studied.

To best take advantage of resource information, a series of questions should be considered when designing an adaptive mobile application. Which adaptation strategy can be best suited to the specific mobile application? What resource information is required? How important is each resource type (or what are the weights of these resources)? What is the relationship between the resource information and adaptation strategy? Studies of these issues can help the adaptation strategies best use the resource information. Choi's work [12] tried to address the third question. This research provides the beginnings of an exploration the relationship between the resource information and adaptation strategy. A discussion regarding the relationship between the adaptation strategies and some of the resources, such as display type, transfer rate and CPU load, can be found in Section 7.2. Further discussions are required.

Dynamic adaptation is the key feature of an adaptive streaming video server.

The topic itself could be a branch of research area. Questions to address include what changing resources to adapt and how frequent to adapt.

To provide better resource-aware adaptation service, some information relating to user preferences and context might be useful. As introduced in Section 3.5, user awareness and context awareness are research areas in and of themselves. However, these three areas have overlaps. In this resource-aware architecture, user input is useful to provide more flexible adaptation service to fit different scenarios. For example, when the network is too slow to transfer a video in a reasonable time period, a user might prefer to quit; another user might prefer to see a poster instead; and some other users might even insist on downloading no matter how long it will take. This type of decisions cannot be determined by the programmer in the design stage.

Location information of the client can help the middleware to predict the coming network situation, and therefore, improving the ability of the awareness of adaptation systems. Other context information such as the devices nearby might affect how to adapt as well. For example, even if the client device is an iPaq, upon detecting a connection to a large wall-sized display, the server will still send the full-size version of requested data instead of a reduced-size version. How to take advantage of user input and context information needs to be studied in the future.

10.4 Conclusions

Although the variety of mobile devices has been increasing, the lack of adaptation service for these devices becomes an emerging problem. This thesis has described a server-based architecture that was designed to support heterogeneous mobile devices under changing resource conditions, including changing network conditions and changing client resource conditions. This architecture uses middleware to provide client resource information, both static and dynamic information, as a service to mobile applications. Appropriate adaptation strategies are implemented by each server application based on this information. Experiments with a simple prototype demonstrate that this architecture can effectively support various mobile devices including a pocket PC, Compaq iPaq. This architecture can be further extended either by moving the middleware or by adding new features.

This research shows that it is possible to provide resource-aware adaptation services for mobile clients. The server-based design reduces maintenance and implementation cost. The middleware in this architecture enables the support of multiple applications by providing adaptation as a service. The responsibility of applying the adaptation strategies is left to server applications. Therefore, the flexibility of making the application-specific adaptation decision based on the resource information is preserved. This architecture is a simple, general, extensible and cost-effective way of making mobile applications adaptive. New resource types, new adaptive server applications and new resource monitoring schemes are easily accommodated with this architecture.

References

[1] Acharya, A., Ranganathan, M., and Saltz, J. Sumatra: a Language for Resource-Aware Mobile Programs. Mobile Object Systems, Lecture Notes in Computer Science. No. 1222, Springer Verlag (D), February 1997, pp. 111-130.

[2] Arlitt, M., and Williamson, C. Internet Web Servers: Workload Characterization and Performance Implications. *IEEE/ACM Transactions on Networking 5* (1997), pp. 631–645.

[3] Aspel, D. Wireless PDA Applications. TRLabs Wireless Networking Workshop, May 2001. Presentation Slides.

[4] Badrinath, B., Fox, A., L.Kleinrock, Popek, G., Reiher, P., and Satyanarayanan, M. A Conceptual Framework for Network and Client Adaptation. *ACM Mobile Networks and Applications 5* (December 2000), pp. 221–231.

[5] Badrinath, B. R., and Welling, G. Event Delivery Abstractions for Mobile Computing. Technical Report LCSR-TR-242, Department of Computer Science, Rutgers University, 1996.

[6] Balakrishnan, H., Seshan, S., Amir, E., and Katz, R. H. Improving TCP/IP Performance over Wireless Networks. In *Proceedings of the First ACM/IEEE Annual International Conference on Mobile Computing and Communications (MobiCom'95)* (Berkeley, CA, November 1995), pp. 2–11.

[7] BlueTooth Research Group. BlueTooth Specification – General Description. http://www. bluetooth.com, February 2001. Product Specification.

[8] Bunt, R. Mobile Computing. Department Seminar, University of Saskatechwan, 2000. Presentation Slides.

[9] Cambridge Computer Corporation. Windows CE File Transfer Protocol (FTP) Client Software Product. http://www.cam.com/vxftp.html, 2001.

[10] CCITT Recommendation T.81. ISO DIS 10918-1 Digital compression and coding of continuous tone still images (JPEG), 1992.

[11] Cheverst, K., Davies, N., Mitchell, K., and Friday, A. Experiences of Developing and Deploying a Context-Aware Tourist Guide: the GUIDE Project. In *Proceedings of MOBICOM' 2000* (Boston, MA, August 2000), pp. 20–31.

[12] Choi, A., and Lutfiyya, H. Delivering Adaptive Web Content Based on Client Computing Resources. In *Proceedings of the 8th IFIP Working Conference on Engineering for Human-Computer Interaction (EHCI2001)* (Toronto, ON, Canada, May 2001), pp. 112–132.

[13] Curran, K., and Parr, G. An Adaptable Framework for Streaming Media to Mobile Applications. In *Proceedings of Wireless 2001* (Calgary, AB, Canada, July 2001), pp. K–3.

[14] Davies, N., Mitchell, K., Cheverst, K., and Blair, G. S. Developing a Context Sensitive Tourist Guide. Technical Report, Computer Science Department, Lancaster University, March 1998.

[15] Demers, A. J., Petersen, K., Terry, M. J. S. D. B., Theimer, M. M., and Welch, B. B. The Bayou Architecture: Support for Data Sharing among Mobile Users. In *Proceedings of Workshop on Mobile Computing Systems and Applications* (Santa Cruz, CA, December 1994), pp. 2–7.

[16] Dolby Laboratories. Advanced Audio Coding. http://www.aac-audio.com, 2000.

[17] Forman, G., and Zahorjan, J. The Challenges of Mobile Computing. *Computer 27*, 4 (April 1994), pp. 38–47.

[18] Fox, A., Goldberg, I., Gribble, S. D., Lee, D. C., Polito, A., and Brewer, E. A. Experience with Top Gun Wingman: a Proxy-Based Graphical Web Browser for the 3COM PalmPilot. In *Proceedings of IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware '98)* (Lake District, UK, September 1998).

[19] Fox, A., Gribble, S. D., Brewer, E. A., and Amir, E. Adapting to Network and Client Variability via On-Demand Dynamic Distillation. In *Proceedings of Seventh Intl. Conf. on Arch. Support for Prog. Lang. and Oper. Sys. (ASPLOS-VII)* (Cambridge, MA, October 1996).

[20] Fox, A., Gribble, S. D., Chawathe, Y., and Brewer, E. A. Adapting to Network and Client Variation Using Active Proxies: Lessons and Perspectives. *IEEE Personal Communications 5*, 4 (August 1998), pp. 10–19.

[21] Froese, K. File Cache Management for Mobile Computing. MSc Thesis, Department of Computer Science, University of Saskatchewan, Saskatoon, SK, August 1997.

[22] Froese, K., and Bunt, R. Cache Management for Mobile File Service. *The Computer Journal 42*, 6 (1999), pp. 442–454.

[23] Gibson Piano Ventures. Baldwin Homepage. http://www.baldwinpiano.com/, 2002.

[24] Heidemann, J., Page, T., Guy, R., and Popek, G. Primarily Disconnected Operation: Experiences with Ficus. Technical Report, Computer Science Department, University of California, Los Angeles, 1992.

[25] Holmquist, L. E., Falk, J., and Wigstrm, J. Supporting Group Collaboration with Inter-Personal Awareness Devices. *Personal Technologies 3*, 1&2 (1999), pp. 13–21.

[26] Housel, B. C., and Lindquist, D. B. WebExpress: A System for Optimizing Web Browsing in a Wireless Environment. In *Proceedings of the second ACM/IEEE Annual International Conference on Mobile Computing and Communications (MobiCom'96)* (New York, NY, November 1996), pp. 108–116.

[27] Howard, J. H., Kazar, M. L., Menees, S. G., Nicholas, D. A., Satyanarayanan, M., Sidebotham, R. N., and West, M. J. Scale and Performance in a Distributed File System. *ACM Transactions on Computer Systems 6*, 1 (February 1988), pp. 51–81.

[28] Imielinski, T., and Badrinath, B. Mobile Wireless Computing: Challenges in Data Management. *Communications of the ACM 37*, 10 (October 1994), pp. 19–28.

[29] ITU-T Recommendation H.263. Video Coding for Low Bit Rate Communication, 1998.

[30] Jiang, Z., and Kleinrock, L. An Adaptive Network Prefetch Scheme. *IEEE Selected Areas in Communications 16*, 3 (April 1998), pp. 358–368.

[31] Jing, J., Helal, A., and Elmagarmid, A. Client-Server Computing in Mobile Environments. *ACM Computing Surveys 31*, 2 (June 1999), pp. 117–157.

[32] Kindberg, T. Mushroom: a Framework for Collaboration and Interaction across the Internet. In *Proceedings of CSCW and the Web, 5th ERCIM Workshop* (Sankt Augustin, Germany, February 1996), U. Busbach, D. Kerr, and K. Sikkel, Eds., pp. 43–53.

[33] Kistler, J., and Satyanarayanan, M. Disconnected Operation in the Coda File System. *ACM Transactions on Computer Systems 10*, 1 (February 1992), pp. 3–25.

[34] Kunz, T., and Omar, S. An Adaptive MP3 Player: Reducing Power Consumption and Increasing Application Performance. In *Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02)* (Hawaii, HI, January 2002), pp. 3843–3852.

[35] Lei, Z., and Georganas, N. D. Context-based Media Adaptation in Pervasive Computing. In *Proceedings of Can. Conf. on Electr. and Comp. Engg (CCECE'2001)* (Toronto, ON, Canada, May 2001).

[36] Liljeberg, M., Helin, H., Kojo, M., and Raatikainen, K. Enhanced Services for World-Wide Web in Mobile WAN Environment. Technical Report C-1996-28, Computer Science Department, University of Helsinki, 1996.

[37] Moving Picture Experts Group. The MPEG Home Page. http://mpeg.telecomitalialab.com/, June 2000.

[38] Noble, B., and Satyanarayanan, M. Experience with Adaptive Mobile Applications in Odyssey. *Mobile Networks and Applications 4*, 4 (1999), pp. 245–254.

[39] Noble, B. D., Price, M., and Satyanarayanan, M. A Programming Interface for Application-Aware Adaptation in Mobile Computing. In *Proceedings of Second USENIX Symposium on Mobile and Location Independent Computing* (Ann Arbor, MI, April 1995), pp. 57–66.

[40] Noble, B. D., Satyanarayanan, M., Narayanan, D., Tilton, J. E., Flinn, J., and Walker, K. Agile Application-Aware Adaptation for Mobility. In *Proceedings of the 16th ACM Symposium on Operating System Principles* (St. Malo, France, October 1997).

[41] Perkins, C. E. Mobile IP. *IEEE Communications Magazine 35*, 5 (May 1997), 84–99.

[42] Satyanarayanan, M. Fundamental Challenges in Mobile Computing. In *Proceedings of Fifteenth ACM Symposium on Principles of Distributed Computing* (Philadelphia, PA, May 1996), pp. 1–7.

[43] Satyanarayanan, M. Mobile Information Access. *IEEE Personal Communications 3*, 1 (February 1996), pp. 26–33.

[44] Schilit, W. *System Architecture for Context-Aware Mobile Computing*. PhD Thesis, Dept. of Computer Science, Columbia University, 1995.

[45] Sorotokin, P. FTP Server in Java. http://home.earthlink.net/~sorotokin/java/ftpd/ftpd.html, January 2000.

[46] Sudame, P., and Badrinath, B. R. Transformer Tunnels: A Framework for Providing Route-Specific Adaptations. In *Proceedings of the USENIX Annual Technical Conference* (New Orleans, LA, June 1998), pp. 191–200.

[47] SUN Microsystems. Java Media Framework API. http://java.sun.com/products/java-media/jmf/index.html, November 2001. Web site.

[48] Tambe, M., Pynadath, D. V., Chauvat, N., Das, A., and Kaminka, G. A. Adaptive Agent Integration Architectures for Heterogeneous Team Members. In *Proceedings of the International Conference on Multiagent Systems* (Boston, MA, 2000), pp. 301–308.

[49] Wade, W. Wireless Networks on Collision Course in the Home. http://www.eetimes.com/story/OEG20000228S0059, February 2000.

[50] WAP Forum. WAP Forum Homepage. http://www.wapforum.org, 1996.

[51] Weiser, M. The Computer for the Twenty-First Century. *Scientific American 265*, 3 (September 1991), pp. 94–104.

[52] Weiser, M. Hot topics: Ubiquitous Computing. *Computer 26*, 10 (October 1993), pp. 71–72.

[53] Weiser, M. Ubiquitous Computing. http://www.ubiq.com/hypertext/weiser/UbiHome.html, March 1996. Web site.

[54] Yarvis, M., Reiher, P., and Popek, G. Conductor: A Framework for Distributed Adaptation. In *Proceedings of 7th Workshop on Hot Topics in Operating Systems (HotOS-VII)* (Rio Rico, AZ, March 1999).

[55] Yeadon, N. J. *Quality of Service Filtering for Multimedia Communications.* PhD Thesis, The Computing Department, Lancaster University, May 1996.