Supporting User Mobility with Web-based Mobile Computing

by

Naizhi Li
B.Eng., Shanghai Jiao Tong University

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE STUDIES
(Department of Computer Science)

we accept this thesis as conforming to the required standard

The University of British Columbia

April 2001

© Naizhi Li, 2001
In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Computer Science

The University of British Columbia
Vancouver, Canada

Date Apr 24th, 2001
Abstract

Recent years have seen the proliferation of mobile computing researches that address the various problems when a device moves from place to place. However, this thesis deals with an additional degree of mobility called user mobility: the ability of users to access their own resources and run their favorite applications, from anywhere using any devices, still in the same way as his own personal computing environment.

A novel computing model called Web-based Mobile Computing (WebMC) is proposed in this thesis to solve the challenges in supporting user mobility such as heterogeneous environment and running local applications using the user’s personal settings.

In this computing model, a user’s personal computing environment is platform-independent and described by an XML document written in a standard vocabulary. Using this document, a web-browser-based, platform-dependent middleware sitting on a WebMC enabled client terminal would enable the user to access his own personal computing environment and run his own favorite applications with his own personal settings. WebMC defines this interface by giving the standard vocabulary.

A prototype middleware has been developed on Linux and Mozilla to prove the feasibility and usability of WebMC. In this prototype, a distinct file system called XIFS is created to provide to web-top applications transparent access of user’s personal environment files, using the XML document as its meta information.
# Table of Contents

ABSTRACT .......................................................................................................................... II

TABLE OF CONTENTS ........................................................................................................ III

LIST OF FIGURES AND TABLES ....................................................................................... VI

ACKNOWLEDGEMENTS ....................................................................................................... VII

CHAPTER 1 INTRODUCTION ............................................................................................. 1

1.1 A SCENARIO FOR USER MOBILITY .............................................................................. 2
1.2 CHALLENGES IN SUPPORTING USER MOBILITY ....................................................... 4
1.3 WEBMC APPROACH ..................................................................................................... 7
1.4 THESIS CONTRIBUTION .............................................................................................. 8
1.5 THESIS ORGANIZATION ............................................................................................ 9

CHAPTER 2 DISCUSSION OF WEB-BASED MOBILE COMPUTING .................................. 10

2.1 THE OVERALL ARCHITECTURE ................................................................................ 11
2.2 WEB-BASED COMPUTING ....................................................................................... 12
   2.2.1 Description of Web-based Computing in WebMC ............................................. 12
   2.2.2 Benefits of Using Web as A Middleware .......................................................... 13
   2.2.3 Rising Levels of Human-computer Interface ...................................................... 14
2.3 MOVING PERSONAL COMPUTING ENVIRONMENT ................................................. 15
   2.3.1 The Structure of Personal Computing Environment ......................................... 15
   2.3.2 Personal Preferences in WebMC ....................................................................... 16
   2.3.3 Moving PCE .................................................................................................... 17
   2.3.4 The Computing Environment Descriptor ......................................................... 17
2.4 LOCAL RESOURCE INTEGRATION ......................................................................... 19
   2.4.1 Using Mimetypes as Representations of Services ............................................ 19
2.5 WEB-TOP APPLICATIONS ....................................................................................... 20
   2.5.1 Web-top Applications as Service Providers ..................................................... 20
   2.5.2 Technologies to Support Web-top Applications .............................................. 21
2.6 THE WEB-BASED MIDDLEWARE ............................................................................ 22
   2.6.1 Accessing User’s PCE Files ............................................................................ 22
   2.6.2 Local Resource Registration .......................................................................... 23
2.7 TRUSTING IN WEBMC .............................................................................................. 23

CHAPTER 3 DESIGN OF COMPUTING ENVIRONMENT DESCRIPTOR .................................. 24
3.1 Requirement Analysis of CED ................................................................. 24
3.2 Using XML in CED ................................................................................. 26
  3.2.1 A Short Introduction to XML .......................................................... 26
  3.2.2 Benefits of Using XML as The Language of CED ......................... 27
3.3 Structure of CED .................................................................................. 28
  3.3.1 Reference and Collection ................................................................. 29
  3.3.2 Translation and Additional Operation .............................................. 30
  3.3.3 Presentation-Oriented Structuring of Collections ......................... 31
  3.3.4 Elements Overview ......................................................................... 33
3.4 Element Details ..................................................................................... 34
  3.4.1 Element CED .................................................................................. 34
  3.4.2 Element UserInfo ............................................................................ 35
  3.4.3 Element RootCollection ................................................................. 35
  3.4.4 Element CollectionInfo .................................................................. 36
  3.4.5 Element URLInfo ........................................................................... 36
  3.4.6 Additional Operation Elements ....................................................... 37
  3.4.7 Element Translation ........................................................................ 38
  3.4.8 Element Self .................................................................................... 38
  3.4.9 Element ReferenceInfo ..................................................................... 38
  3.4.11 Element ServicePool ...................................................................... 39
  3.4.12 Element PreferencePool ................................................................. 39
3.5 Summary ............................................................................................... 40

CHAPTER 4 PROTOTYPE DESIGN AND IMPLEMENTATION .......... 41
4.1 Prototype Background and Requirements ........................................ 41
4.2 Prototype Platform ................................................................................ 43
4.3 Implementation Background ............................................................... 43
  4.3.1 Linux Virtual File System ............................................................... 43
  4.3.2 The Mozilla and Plug-in Architecture ......................................... 44
4.4 Prototype Architecture ....................................................................... 45
  4.4.1 PCE File Access Service Provided via XIFS .................................... 45
  4.4.2 Providing Application Registration Service via Universal Plug-in and Application Description Table .................................................. 46
  4.4.3 Components Overview ................................................................. 47
4.5 XIFS Redirector .................................................................................... 49
4.6 XIFS Processor ..................................................................................... 52
4.7 The Application Handler .................................................................... 58
4.8 Summary ............................................................................................... 58

CHAPTER 5 RELATED WORKS ................................................................. 60
5.1 Terminal Mobility Systems ................................................................. 60
  5.1.1 Odyssey ......................................................................................... 62
  5.1.2 Rover ............................................................................................. 63
5.2 Universal Personal Computing ............................................................ 64
5.3 Agent-Based Systems ......................................................................... 66
5.4 RESOURCE DISCOVERY SYSTEMS .............................................. 67

CHAPTER 6 CONCLUSION AND FUTURE WORK .......................... 69

6.1 SUMMARY ................................................................. 70
6.2 FUTURE WORK .......................................................... 71
   6.2.1 System Improvements ............................................. 71
   6.2.2 New Directions ..................................................... 72
6.3 CONCLUSION ............................................................. 72

REFERENCES: ......................................................................... 73

APPENDIX A: THE COMPLETE DTD OF CED .............................. 76
List of Figures and Tables

Figure 1.1 A Scenario Fitting to User Mobility...........................3
Figure 2.1 The overall architecture of WebMC ..........................12
Figure 2.2 The position of the web browser in WebMC model ..........14
Figure 2.3 Evolution of Human-Computer Interface ......................15
Figure 2.4 Position of CED in WebMC Systems..........................19
Figure 3.1 Structure of CED .............................................33
Figure 3.2 The Structure of Collections in CED ..........................35
Figure 4.1 Linux Virtual File System .....................................46
Figure 4.2 The Software Architecture of The Prototype Middleware .......50
Figure 4.3 The XIFS Redirector ...........................................53
Figure 4.4 Header Structure of Protocol between XIFS Redirector and XIFS Processor .........................................................53
Figure 4.5 Data Structure of The Nodes in The Internal Tree ..........57
Figure 6.1 Range of Adaptation Strategies ...............................64
Figure 6.2 Architecture of UPC ..........................................67

Table 4.1 .............................................................................54
Acknowledgements

A lot of thanks go to my supervisor, Dr. Son Vuong. This thesis would not be possible without his supervision and encouragement. I learned a lot from Dr. Vuong in my years in UBC. Also thank Dr. Alan Wagner to be my second reader.

Thanks go to my friends at UBC. I had great fun with you guys when I was having breaks from the reading, programming and writing.

NAIZHI LI

The University of British Columbia

Apr 2001
Chapter 1

Introduction

Recent years have seen the fast development of wireless technologies that engenders a new paradigm of computing—mobile computing. The common definition of mobile computing is that users who carry portable devices have access to information services through a shared infrastructure, regardless of their physical location or movement behavior. From this definition, we can see that it assumes the fixed association between a mobile user and a certain mobile device.

However, this association may not always exist in the real world. For instance, not all users own a PDA or notebook. And if a user owns a mobile device and carries it with him/her, the inherent limitations caused by its mobile nature may prevent it from providing all the services that a stationary computer with wired Internet connection could provide. It’s desirable that computers be installed in hotels, airports to support mobile users the maximum possible services. In this case, the association between the mobile user and the certain mobile device no longer exists: the mobile user is now free to choose any terminal for the services he needs.
We call this additional degree of mobility *user mobility*: the ability of users to access defined services from any terminal in the network (including user’s mobile devices) while maintaining their personal computing environment. The original mobile computing, in which it is the device that’s mobile and the user is attached to the device, is called terminal mobility for comparison reason.

A lot of research has been carried out on the subject of supporting terminal mobility, like Coda and Odyssey from CMU, Rover from MIT and Bayou from Xerox PARC. These research projects center around the concept of *adaptation*, where applications, systems, or both dynamically adjust their behavior to the limited, variable and changing mobile resources.

However, only a few works has been done exploring the concept of user mobility. Often called *personal mobility*, most of the research projects focus on the user mobility of one specific application. In this thesis, we introduce a new model called WebMC, trying to provide a system solution to the challenges and problems in supporting user mobility.

This chapter introduces the thesis to the readers in the following way: first we motivate this thesis by presenting a scenario about user mobility. The challenges in mobile computing are then explained and the problems the thesis is trying to resolve are outlined. After that our WebMC approach is briefed and finally thesis contribution and organization are given.

### 1.1 A Scenario for User Mobility

Consider the following scenario illustrated in Figure 1.1.

*A scholar has a personal computer in his office. He uses this computer everyday and he is happy with the software installed in this computer and services like printing, scanning and word processing. The settings of the software environment have been changed to satisfy his special needs. He’s familiar with his own settings and he knows where to find the services he...*
Figure 1.1 A scenario fitting to user mobility needs. In one sentence, he can work very efficiently with this hardware and software environment.

He travels to another city for a conference and he’s staying in a nice hotel. He runs WebMC on his notebook connected to the Internet via a wireless link. In the “Personal Documents” page of WebMC, he would find a link “Presentation XYZ” and click on that link. The notebook will automatically download the document and he can resume the preparation of the presentation using the corresponding application as though he was working in his office. He never needs to worry about synchronization and FTP.

During preparing the presentation, he finds out that he needs to discuss with his colleagues and students. He wants to setup a video conference with them, which requires far more network bandwidth than what he can get from the wireless Internet connection of his notebook. He turns to the hotel and the hotel provides him with a terminal with high-speed Internet connection, that has WebMC installed. In the “Services” page he would find his video conferencing software and he could setup the conference in exactly the way he’s familiar with. The conference software will work the same way as when he holds a video conference in his office.

This scenario is only one of the similar situations where a user’s mobile device may or may not be sufficient to meet the user’s needs. In these situations, we
should focuses more on the mobility of the user itself, rather than simply associating the user with the fixed mobile device and enable the mobility of the associated device.

On one hand, there are many real world restrictions that prevent users to maintain the association with his or her mobile device. For instance, a user may simply think a notebook is too heavy to carry to everywhere. The battery of a specific mobile device may run out any time.

On the other hand, in a computing environment that has fixed association between a mobile user and a mobile device, there are some inherent resources constraints. The fixed association places some requirements like lower weight and smaller size on the mobile device. And because of these requirements, a mobile device is limited in local resources such as disk size, physical memory and networking performance. These constraints are inherent to mobile devices, because technological advances that increase the resources available to a typical mobile device will also be applied on desktop devices. Another restriction lies in the fact that most mobile devices rely on wireless networks for connectivity. Such networks tend to have lower bandwidth, longer latencies and variance from area to area.

With this scenario in mind, we set the goal of this thesis to answering the following questions: how are we going to achieve user mobility? To answer this question, we first have to understand the biggest challenges ahead of us in supporting user mobility. The next section will try to analyze these challenges.

1.2 Challenges in Supporting User Mobility

While recent technologies such as Mobile IP and CDMA have somehow enabled Internet connection for mobile devices in the network layer, the concepts of mobile computing and user mobility introduce new technical challenges in the higher layers. Employing these two concepts changes the environment dramatically, and simply reusing whatever we have today in an immobile
computing environment is insufficient and often troublesome under mobile circumstances.

For example, the network bandwidth constraints on a mobile device are considered one of the most important challenges by many researchers [Katz1994] [Noble1998] [Satya1995]. Furthermore, with the introduction of user mobility, a number of user mobility specific challenges have emerged. In the later part of this subsection, we will discuss the challenges that are only specific to supporting user mobility. We put the challenges into several categories:

Supporting user mobility means heterogeneous environment. Terminals provided in hotels or airports may be constructed on different hardware architectures and may run different operating systems. Users must be able to see and use effectively their own computing environment from these heterogeneous terminals without even knowing the underlying hardware architecture and operating system. When they load in their computing environment in a different terminal, they should see the same or similar screen – and follow the same actions to achieve the same goal.

This heterogeneity calls for a cross-platform representation of a user’s computing environment. This platform-independent representation, on the other hand, must fit to different kinds of hardware and software platforms that follow WebMC model. Certain abstractions, both on the usage of heterogeneous computing resources and on the representation of a user’s computing environment must be drawn.

Supporting user mobility means the integration of local resources into a user’s computing environment. A user’s computing environment relies on a set of resources, either hardware or software, which enable the services the user needs. For instance, to obtain a hard copy of a certain document, you must have a printer to print it out, and a software application that understands the document and translates the document into a printer-friendly representation.

Resources could be divided into two categories: hardware resources and software resources. Hardware resources means hardware facilities that can be operated by a
computer provided to the users. Examples of hardware resources include printers, scanners, hard disk spaces, projectors and so on. When you go to a place, you can’t simply bring all of them with you; you have to use locally available hardware resources. A user should be able to use one kind of hardware just like how he/she uses the same kind of hardware in his/her office.

Software resources are applications that provide certain services to a user. When working on a different terminal, a user may want to use the same or similar software applications to get the same services. These locally available software applications must be made available to the users.

New networking and software technologies have, to a certain degree, given applications the ability to migrate through Internet [Jini2000]. In some cases the applications could be running remotely or be downloaded to local machine when necessary. For example, web technology allows information exchange between a client and a web server. Java and ActiveX technologies make it possible for codes to be downloaded to local machine on run-time. However, these technologies could not solve all the problems. Due to web’s coarse-grained download/upload nature of the Hyper-Text Transfer Protocol, traditional client/server model implemented as GUI programs works better for applications with intensive interaction [Nielsen1999]. In the case of ActiveX and Java applets, the concerns of efficiency, security and most importantly, download speeds prevent them from being extensively used. It is hard to imagine having to download a huge application each time before we want to use it.

Therefore computing environments must be integrated with local resources to enable users to access and use them. Under some circumstances, a local environment may not be able to satisfy all the needs of a specific user due to the lack of available resources. However, a system supporting user mobility should be able to provide the user with the resources the user needs to a maximum possibility.
Supporting user mobility asks for a security model. Security in supporting user mobility has two senses of meaning: protection of local resources from being accessed by unauthorized users, and protection of users' personal files to be modified by terminals.

For example, the authorities have to ensure the terminals provided in an airport in a stable state and don’t get corrupted, damaged or hacked by users’ mistaken operations or intentional disruptions. On the other hand, a user certainly does not want his own personal information like e-mails to be accessed by an untrusted organization.

Finally, supporting user mobility requires some degree of adaptation. The differences in the ability of different terminals/mobile devices will certainly affect the user in many aspects. For instance, an Ethernet connected computer may enable the user to download fancier interfaces for his computing environment while a wireless connection may restrict the interface to be text only. In this case, the ability of the system and applications to adapt to the changed environment will be very desirable. However, there has been a lot of research work done in this area [Noble2000] [Joseph1997] [Jing1999].

1.3 WebMC Approach

The widespread use of the World Wide Web has led us to consider applying web-based environment to achieve the goal of supporting user-mobility. The following characteristics of such an environment are prevalent: i) The web is a widely accepted technology. Almost everyone knows how to browse web pages. ii) The web is a cross-platform design environment.

Thus, we propose to use a web browser as a middleware to support user mobility – so called Web-based Mobile Computing. Different platforms have different middleware, but all middlewares have the same northbound interface to access the user’s computing environment and display the computing environment to the user in a pre-defined way. Extensive Markup Language (XML) is used as the core of
this interface. In this case, cross-platform of the interface is achieved because the language itself is neutral.

The goal of the WebMC middleware is to satisfy the following conditions: i) retain user’s personal computing environment; ii) support the platform-independent XML interface defined in the WebMC computing model; iii) provide services to the user at a maximum possible level with ease and efficiency.

1.4 Thesis Contribution

This thesis has two major contributions: proposes a web-based structure as a system solution to support user mobility; and proposes the structure of the core XML structure. Specifically, it is constructed by presenting the following contributions:

- This thesis first argues that terminal mobility is insufficient under many circumstances. Under those circumstances, we should provide users with user mobility support.

- This thesis outlines the challenges in supporting user mobility. These outlined challenges define the problems of this thesis and serve as the direction in many places throughout the thesis.

- This thesis proposes a novel computing model called WebMC. This computing model, based on web structures, tries to solve the challenges and problems defined earlier. The basic concepts of this computing model are then discussed.

- This thesis then proposes a generic, cross-platform design of computing environment descriptor. Using the language of XML, it defines the format of any computing environment descriptor by using a Document Type Definition.

- The functionality of the WebMC middleware is discussed in depth. It argues that “accesses to the user’s computing environment files” and “local resources
registration and management” are the most important functions that a middleware should support.

- This thesis presents the design and implementation of a prototype system of WebMC computing model. It argues that this prototype proves the feasibility of WebMC computing model.

- After giving some example applications, this thesis draws the conclusion that WebMC computing model is feasible and usable, and has many advantages over terminal supporting environments. Future work directions are also discussed.

1.5 Thesis Organization

The rest of this thesis comprises 7 chapters. Chapter 2 presents the concepts of WebMC in the form of discussing the key points for WebMC to support user mobility. This chapter gives the foundation of the following design of computing environment descriptor and design of them WebMC prototype.

Chapter 3 gives the design of the Computing Environment Descriptor, or CED. CEDs are written in XML and this chapter gives a Document Type Definition to define the vocabulary and format of any CED.

Chapter 4 gives the design and implementation of the WebMC prototype. Design requirements are outlined and the architecture is given to satisfy the requirements. Internal structures, modules and communication protocols are also presented.

Chapter 5 briefs the related work of WebMC. This chapter presents various projects from universities and companies to compare to WebMC. They include Odyssey from CMU, Rover from MIT, UPC from UBC, NetChaser from U of Catania, Italy and Jini from Sun Microsystems.

Chapter 6 finalizes this thesis by giving the conclusion comments and future work directions.
Chapter 2

Discussion of Web-based Mobile Computing

The previous chapter argues that terminal mobility is insufficient under many circumstances and user mobility should be supported in that situation. It also outlined the challenges which are heterogeneous environment, integration of local resources, a new security model and adaptation, to support user mobility.

In this chapter, the WebMC (Web-based Mobile Computing) computing model is proposed to address these challenges. It is called a computing model because it is rather a good way of doing things than an actual program that enables users to fulfil certain tasks. This computing model defines the basic structure, the description of a user's computing environment, and the way the middleware handles the user's computing environment.

A prototype is implemented and presented in detail in later chapters. However, the goal of the prototype is to prove the feasibility of this computing model; it does not necessarily define the way middlewares should be implemented.

This chapter tries to present the concepts behind WebMC – how WebMC came to our mind and why it's designed this way. This chapter is organized in the
following way. First the overall architecture is presented. After that we argues for the benefits of web-based computing. The personal computing environment is then discussed in depth. The central part of the personal computing environment – the computing environment descriptor is then defined and discussed. The structure of the middleware is then presented. And at last there is a short discussion on the trusting and security.

2.1 The Overall Architecture

A WebMC client refers to a mobile device or a fixed terminal that provides services to the user directly. It must have certain computing capability (CPU & memory), be able to connect to the Internet, be installed with a specially enhanced web browser, and follow the WebMC computing model. Different WebMC clients are able to provide, to the maximum possibility, the same user with the same pre-defined services that the user needs.

A WebMC client needs a home server to get user-related information such as the user’s preferences of an application. Because the client is based on web, the home server should be implemented in a machine running a server program (e.g. a www server). A specific user’s user-related information is stored in his/her home server.

*Figure 2.1 The overall architecture of WebMC*
and transferred through the Internet to the client when necessary. A home server can serve multiple users simultaneously.

The overall architecture is shown in Figure 2.1.

Depending on a specific user’s situation, the home server may be just a standard web server with HTTP/1.1 support where no CGI script or other modifications is required on the server. In this special case, only HTTP is used as the communication protocol and information flow from the client to the server could be implemented as PUT, POST, UPDATE and DELETE methods.

2.2 Web-based Computing

2.2.1 Description of Web-based Computing in WebMC

In a traditional operating system such as UNIX, Microsoft Windows and MacOS, the human-computer interface consists of a shell program (sh/csh/Windows Explorer) plus a set of tools. The shell programs and the sets of tools are operating system dependent and they differ dramatically among different operating systems. For instance, UNIX is considered mainly command-line based while MacOS is considered to be mainly GUI based. Thus a user has to learn how to use them before they are able to use them effectively.

WebMC differs from the traditional way in that the web browser is a middleware sitting on top of the operating system, and under other applications. This web browser middleware is itself the human-computer interface. The style and content of the user interface is determined by a set of web pages, instead of the hard coded buttons, icons and windows in a traditional operating system.

Any resources, applications and other services must be accessed through the web-browser interface. For instance, a user may have several personal documents in different formats that need different kinds of applications to read and process. It would seem natural to have a web page with links pointing to those documents. If the user wants to access any of his/her documents, s/he simply goes to that page, clicks on the specific link and the computer will pick up the right application,
possibly downloaded automatically from service providers or even the user’s home server, to process the document.

Figure 2.2 showed the position of the web browser in WebMC model. The human-computer interface is above the web-browser based middleware.

2.2.2 Benefits of Using Web as A Middleware

Using web as a middleware has certain benefits over the traditional model. First, *web-based system is platform independent*. In WebMC, it is not the operating system that determines the user interface. Instead, it is a set of web pages, called Computing Environment Descriptor, that determine the user interface. Because web pages are just text files, it is easier for a user’s computing environment to migrate from machine to machine. A user will see the same thing if s/he have these same pages, no matter what kind of hardware and operating system s/he uses and no matter where s/he is.

*Web-based system hides hardware, operating system and software details from users.* The browser serves as an intermediate layer between these system details and the user. A user interacts with the browser only, and thus does not need to know what lies beneath the browser. For example, users don’t even have to know there is an underlying file system, just like today’s OS (Unix or Windows) users don’t have to know about device drivers.

Finally, *web-based system is very flexible in describing different user interfaces*. With the increasing descriptive power of web languages such as CSS, users will
be able to choose their desktop from a big reservoir of styles. For example, [WebOS2000] has shown that XML and CSS/XSL technologies could describe a windows style desktop.

These benefits are essential to supporting of user mobility. As we mentioned in the previous chapter, supporting user mobility require operating on heterogeneous environment. The cross-platform nature of web meets this requirement and also provide flexibility in the interface design.

2.2.3 Rising Levels of Human-computer Interface

Generally, we have raised the human-computer interface to a higher level in the software hierarchy. Figure 2.3 has shown the evolution of human-computer interface and the position of WebMC. In the history of software development, we can see a trend of rising level of human-computer interface. From the very first card-reader to the command line operating system, and then to GUI version of Unix and Windows. In each step of raising the human-computer interface, we

![Figure 2.3 Evolution of Human-Computer Interface](image)

Figure 2.3 Evolution of Human-Computer Interface

gained some ease of use and more functionality, and lost some flexibility at the
same time. This holds true for the WebMC computing model as well. In the last step to WebMC, the user mobility is achieved using web-based technologies.

2.3 Moving Personal Computing Environment

The main goal of WebMC is to retain a user's personal computing environment (PCE) no matter where the user may be and what kind of computing devices he is using. This is part of supporting user mobility. Thus, PCE is a very important concept in WebMC. In WebMC, a user's PCE is constituted of a set of files and documents with and description meta data containing the structure of the PCE.

2.3.1 The Structure of Personal Computing Environment

A user's PCE should contain all the information that a user needs to re-establish his computing environment at any terminals. The information should include the following:

- PCE should contain the user's personal documents. Personal documents are important components of a user's computing environment. Users need to work with them, store result and temporary result in them, and present them in a proper way.

- PCE should contain links to a user's favourite applications and services. Users should be able to invoke his/her favourite applications from this computing environment from any WebMC-enabled terminals.

- PCE should contain personal preferences for different kind of services and applications. Different users may have different settings for the same application. Any terminals supporting WebMC must be able to restore the settings of different users for the same application.

- PCE should contain style information to present its structure in a platform-independent and machine-understandable way. A WebMC client machine should be able to understand, access and modify a user's
computing environment. In WebMC, we use an XML document to describe the whole PCE.

However, PCE should not contain the binary code and libraries of heavy weight services. As stated in the previous chapter, the application size is not possible to transfer over the Internet, and there are other issues like installation process and etc. These applications and services should be considered local resources/service providers and should be provided by the local machine.

Most of the information in a PCE appears as a file, which shares the same abstraction in a file system, i.e. a stream of unstructured bytes. These basic units are structured by a special document in PCE.

PCE could be stored in the home server like a Unix home directory or an NT user profile. All the configuration files and personal documents are stored under the root of a user’s PCE. How these files are structured under the PCE’s root depends on each user and his applications and is customizable by the user. For instance, all files pertaining to a certain application may be placed under a sub-directory dedicated to that application.

2.3.2 Personal Preferences in WebMC

Personal preferences for applications are an important part of a user’s personal computing environment. It is the WebMC system’s responsibility to restore the user’s personal preferences without the user’s intervention. A user shouldn’t be required to know which files keep his preferences.

Some applications already have the ability to access the personal profile from the network. For instance, Netscape Communicator is able to download and update some preference files from a web server or a LDAP server. However, most applications still use local files to store user’s preference. From a system point of view, it is also desirable to abstract downloading of preferences to be a system service provided to all applications.

There are generally two ways to provide such a system service:
• Provide a certain interface and applications can register the location of the personal preference files through this interface. An example is the Windows registry file, where applications register its existence and a lot of other common information like path, DLL dependency and etc. The Computing Environment Descriptor (refer to section 2.3.4) is a good place to store the information.

• Provide a home directory (similar to a Unix home directory) for each user. All the information relating to a specific user, including his/her personal preferences, is stored in the user's home directory. Thus when the applications access the preferences files, the system will be able to download the files from the user's home server automatically.

2.3.3 Moving PCE

In WebMC, a user's PCE is stored in the home server and downloaded from the server to the client when necessary. A PCE consists of many files (personal documents, personal preferences and etc.), and only those files accessed should be downloaded and cached.

In this way, a user's PCE could be considered to move with the user, no matter where s/he goes as long as s/he has Internet connection.

2.3.4 The Computing Environment Descriptor

As we mentioned in section 2.3.1, a PCE should be able to present itself to a WebMC client in a structured, platform-independent and machine-understandable way. In WebMC, we use an XML document to describe a user's PCE. This XML document is called Computing Environment Descriptor or CED.

Basically, CED contains the meta data of a certain PCE. For example, a CED could give the URLs of all the personal documents a user has. When the user wants to load a document, the system could simply download the document using the given URL. The links to the user's favourite applications and services should also go into this document.
All CEDs are written in a pre-defined set of vocabulary. This vocabulary will be discussed in chapter 3. Different users have different contents inside their PCE and CED, but they all speak the same vocabulary so that any WebMC client is able to understand all the CEDs. A WebMC client should be able to access and modify relevant items in a CED when necessary.

CEDs should be represented to the user using web-based languages like XSL [XSL2000]. XSL retrieve information from XML documents and generates web pages using the information. Each of the XSL documents is a style of representation of the CED content. The generated pages are actually the user's desktop. Figure 2.4 shows the position of a CED in a WebMC system.

These web pages and how they are organized together differ from user to user. For instance, a user may prefer multiple pages with relatively smaller page size while some other may prefer a big page with everything on it. Some users may even have multiple representations of his/her CED for different purposes.

With the increasing descriptive power of web languages like XSL, users will be able to choose a big reservoir of styles no matter what is inside their CED. For example, [WebOS2000] has shown that CSS and other technologies could

Figure 2.4 Position of CED in WebMC System
describe a Windows style desktop.

Since CEDs are written in XML and represented using web-based languages like XSL, they are cross-platform. Regardless of the kind of hardware and software one uses, one can access one’s CED as long as the web is supported. A standard XML document is used to store certain information, and the pages later retrieve the information from this file.

CED along with the XSL representation can be viewed as a counterpart of the SHELL program in an operating system. It describes the user’s desktop and how user accesses various kinds of resources and services. In WebMC, a desktop is not determined by the kind of operating system one uses; it is rather up to the user and the kind of desktop the user wants.

2.4 Local Resource Integration

As we mentioned in chapter 1, local resource integration is very important to mobile computing. Local resources can be divided into two categories: hardware resources and software resources. It is natural that users make use of local hardware resources as much as possible since a user won’t be able to carry all the hardware resources that he needs with him. In the mean time, locally available software should be used as much as possible because of download speed and many other constraints.

The question remains: how can we integrate these local resources into a user’s PCE? When a user goes to a computer, how can s/he make the best use of the local resources of this computer through his own familiar interface? In this section, we’ll give the answer.

2.4.1 Using Mimetypec Types as Representations of Services

In WebMC we use a strong type mechanism for applications and documents. Each of the applications is related to one or more specific mime types. Each document also has a mime type and it is processed by an application capable of this kind of mime type. Thus, a mime type represents a certain kind of service. A
document will need a special service to process and fulfil certain tasks the user needs.

Mimetypes serve as a bridge between local resources and user’s computing environment. User’s favourite applications is represented by mime-types and the system will choose the right application when that mimetype is hit.

A link to a document with a certain mime type could be a way to invoke the corresponding application. For example, we can have a special web page that contains a “new Word Document” link pointing to a blank Word Document. The link can serve as a method to invoke Word application. Compared to a file in UNIX or Windows system, the mimetype of a document is much stricter than file extension names, although both of them denote the type of that file. An OBJECT tag in an HTML with mimetype in the tag would also serve as a way to invoke a certain document.

This mimetype scheme calls for standard definitions of mimetypes all over the world. Fortunately, a lot of mimetypes have already been defined and are well in use today. We also look forward to see a more standard mimetype definitions in mimetypes.

2.5 Web-top Applications

2.5.1 Web-top Applications as Service Providers

Applications are very important resources to users. In WebMC, all other applications are invoked and controlled by the web browser, thus they can be considered sitting on the top of web, so called web-top applications.

In a WebMC client, services are defined by mimetypes and provided by web-top applications. When a user need a service, s/he will tell the system what service s/he needs by specifying the mimetype of the service. The WebMC client should then pick up the right application that supports this mimetype and invoke the application.
2.5.2 Technologies to Support Web-top Applications

There are currently several existing technologies that would enable web-top applications: i) helper applications; ii) plug-ins; iii) Java applets and iv) ActiveX control. The common feature of these technologies is that they all associate an application with web browsing.

A helper application is a normal application that is known to the web browser through the help of the operating system (e.g. the Windows Registry) and that is able to process documents with certain mimetypes. On the other hand, a helper application is not aware of the existence of the browser. Hence, the only way the web browser can communicate with it is via a temporary file.

A plug-in is a dynamic library that works with the web browser interacting via a interface defined by the web browser. Plug-ins can communicate with the browser through the methods defined in the interface, thus they are more flexible. The web browser comes to know the related information of a plug-in through the interface too. It is a convenient way to implement web-top applications and many contemporary applications comes with a plug-in version.

An applet and an ActiveX control is a piece of downloaded code to be run on the client’s machine. An applet is written in Java and cross-platform while an ActiveX control is written in binary code (x86) and only run on a specific platform (Windows). However, it makes better use of the native environment.

Meanwhile, on-going research has suggested even more possibilities to support web-top applications. Among them are Jini [Jini2000], Distributed COM [DCOM1998], Service Discovery [Czer1999] and etc. The development of application migrating technology will also give WebMC clients a broader range of available applications to support user’s needs.

With these future possibilities in mind, web-top applications could be divided into two categories: local or dynamic. A dynamic application is (partially) downloaded when necessary while a local application is pre-installed well ahead of usage. If a
corresponding local application is not available to open a specific document, the user may choose to either cancel the operation or revert to a dynamic application. The smaller applications could be desirably implemented as dynamic applications that are downloaded to the client only when necessary. This could give mobile devices greater functionality without using expensive mobile storage resources.

2.6 The Web-based Middleware

In WebMC model, the web browser serves as a middleware that sits on top of the operating system and under web-top applications. Nevertheless, to support user mobility, this web browser based middleware should be enhanced to provide certain services to web-top applications above it. They are i) access to a user’s personal environment files; and ii) application registration and management.

Beside the two services, the middleware should also support session control function (i.e. login and logout). The middleware should identify the loading of a user’s CED as the start of a session, and ensures saving back the user’s CED at the end of the session. The middleware should also ensure the consistency of user’s personal computing environments under situations like crashes and interrupted connection.

2.6.1 Accessing User’s PCE Files

In WebMC, a user’s computing environment moves with the user. When a user goes to a public terminal and start an application through the web interface, the application will access the user’s personal environment files in two cases: i) to access the user’s preference files for this application, and ii) to operate on a specific personal document.

The web-top applications rely on the middleware to access user’s PCE files. Some current applications such as Netscape 4.7x have the ability to store personal preference files in a remote HTTP or LDAP server. WebMC chooses to provide this function as a system service rather than relying on the applications themselves.
2.6.2 Local Resource Registration

When a user clicks on a link to a document with a specific mimetype, the web-based middleware should invoke the corresponding application to process this document. This requires the application to register its existence and some parameters (e.g. what mimetypes it can handle) to the middleware. In other words, the middleware should provide certain ways to register and manage local application for future use.

2.7 Trusting in WebMC

The security model of WebMC consists of two aspects: protection of the local resources from unauthorized user access; and protection of user’s files from unauthorized accesses by the terminals or the applications running on the terminals.

There are basically two ways of security: sandboxing and trusting. Sandboxing is used in the situation where you have to run many untrusted programs where certain operations are not allowed. For example, a Java applet running in a browser is not allowed to access local files or communicate with other servers. In WebMC, sandboxing is suitable to protect local resources from unauthorized user access. The system should know what kind of local resources could be accessed and what could not.

We propose using trusting and certificates to protect a user from unauthorized access of his personal computing environment files. The home server transfers the user’s personal files to the WebMC client only after the client has been verified to be a certified WebMC client. This process needs a central certificate authority like Verisign. A user’s computing environment is encrypted before transferred to the WebMC client and only this client can decrypt this environment, thus proving its identity to the user.
Chapter 3

Design of Computing Environment Descriptor

The WebMC computing model requires applying a user’s Personal Computing Environment on a WebMC client. User’s PCE moves from place to place, thus the interaction between the PCE and the WebMC clients must be in a pre-defined, standard way. In WebMC, the structure of a user’s PCE is maintained using an XML file, or so-called Computing Environment Descriptor (CED). A WebMC client accesses documents in user’s PCEs via their CEDs. All CEDs must use the same vocabulary to make sure WebMC clients understands them.

This chapter presents the design of CED. First, the functionality and requirements of CED are discussed. After that, we discuss why we choose XML to be the language of CED. Then the structure of CED is discussed and the elements are presented in detail. Finally, some issues in the design of CED are summarized.

3.1 Requirement Analysis of CED

Computing Environment Descriptor, or CED, plays a very important role in the Web-based Mobile Computing Model. In general, CED serves as a bridge between a user’s PCE and the middleware/web-top applications running on a WebMC client. On one hand, the CED is a structured XML document that
describes all the contents of a user’s PCE – in other words, it is the metadata of the PCE; on the other hand, it is written in a standard vocabulary that WebMC clients understand, thus enabling the clients to locate, access and (possibly) modify the relevant information in the user’s PCE. The WebMC middleware in turn provides a system service for web-top applications to access PCE files and documents.

From the above discussion, we can see there are two basic functions that a CED must provide: describe the PCE and allow any WebMC client to retrieve necessary meta information from it. Obviously the CEDs must be written according to a standard, to enable WebMC clients built on top of different hardware and software platforms to understand them.

Besides these two basic functions, there are some other requirements that a CED should follow. The first one is that CED must be platform-independent. This is quite obvious since a CED moves from machine to machine and these machines may have different software or hardware architecture. Furthermore, the design of CED should not bear any implications of any specific system. For example, a CED should not assume the strict tree structure of a filesystem as on some systems symbolic links could destroy this structure.

The design of CED should enable different WebMC middlewares to provide system services to web-top applications, especially, mapping to a file system. As analyzed in Chapter 2, WebMC middleware should provide two kinds of services to web-top applications: access to user’s computing environment documents, and local resource registration and management.

Many web-top applications access the personal documents and preference files through a file system service provided by operating systems. For example, Microsoft Office stores personal information under Profile\Application Data\Microsoft\Office directory. Thus it is very possible that a WebMC middleware provide file system style support to help web-top applications access
PCE documents. The CED design should incorporate considerations to map PCE documents to a file system.

In a sense, this is conflicting with the platform-independent requirement because a truly platform-independent CED design should bear no implications from a file system. The design of CED should achieve a balance between these two requirements.

The design of CED should facilitate flexible ways to be presented to users. Since all CEDs fit to the same standard, they may all have similar structures. However, different users may have very different requirement on how the CEDs are represented to them. For example, many users may prefer Windows-style Start Menu while others may prefer icons on the desktop. CEDs should be able to support as many different styles as possible, possibly through the help of the WebMC clients.

CED should provide certain degree of interaction with WebMC clients. A WebMC client should be able to manipulate the meta data as well as the normal data in a PCE. For example, a personal document may be deleted from the user's PCE and this will cause some update to the CED – the meta data of PCE. The design of CED should allow updates as well as accesses of a specific item in CED.

### 3.2 Using XML in CED

With these requirements in mind, it has been decided to use XML as the underlying language that CED should be written with. This section will present the reasoning behind this decision.

#### 3.2.1 A Short Introduction to XML

XML [XML1999], originally coming from SGML, is to provide a structured, text-based way of storage of data units. XML is a set of rules, guidelines and conventions for storing information in plain text that is common to all Internet and easily understood by computer programs.
An XML document contains one or more data units that are called elements. XML makes use of tags to delimit the scope of an element, i.e. the content between the start-tag and end-tag. Like tags in HTML, a start-tag is a name and a set of attributes with "<" and ">" between them and an end-tag is a tag with a "/" before the name. So an XML element may look like this

```xml
<eName aAttribute="aValue">
    ...content...
</eName>
```

This resembles the format of HTML. However, in HTML some tags don’t have a corresponding end-tag, while an XML element always has an end-tag.

Different from HTML, an XML document does not have a pre-defined set of tags. It rather leaves the job of interpreting the meanings of the tags to two other kinds of document: Document Type Definition (DTD), and Schema. The DTD or Schema defines the structure and meaning of elements and one can validate an XML document against the DTD or Schema.

### 3.2.2 Benefits of Using XML as The Language of CED

As one of the web-based languages, XML comes with no dependency on any platform. XML ensures platform-independence in the lower level (e.g. data formats). However, the structure of a CED should still be designed to avoid introducing any platform dependence in the higher level.

XML is a well-formed language and is equipped with formal vocabulary definition languages like Document Type Definition and Schema, making it easier for the web-based middleware on WebMC clients to interact with CEDs. A computer program can interpret the semantic of a specific XML element by looking at the name of its tag. For example, in an article description XML, an element with a tag `<Paragraph>` probably means that the content in the element forms a paragraph of an article. Other web languages, such as HTML, are not suitable to be understood by computer programs. In HTML, most of the pre-
defined tags only have a semantic for displaying them on the screen. This is a huge advantage of XML over other web-based languages like HTML because it makes XML "meaningful" to programs.

XML enables the separation of content and style. Content could be kept in XML files while styles could be in other kinds of files like CSS and XSL. This separation makes it easier to support flexible presentation of CEDs to users. For example, there could be a pool of styles to present the same content, and a user could just browse all the styles and pick up whatever s/he likes best.

XML has a tree structure. An element must be either wholly contained in another element, or be separate. An element must be outside. This strict structure is flexible enough to be able to describe all the possible PCEs.

Furthermore, a set of closely related technologies helps XML to have better functionality. For example, Extensible Stylesheet Language, which helps present an XML document to the user using the style defined in the style and the content from the XML document. XSL is the preferred technology for presenting CED to users. Other examples include Xlink, XPointer, DOM and XPath. With these technologies and emerging open source code for these technologies, it will be very easy for a programmer to write XML based programs.

Last, XML is an accepted technology, widely used in many aspects. The maturity and adoption of XML means more support, better documentation and more tools.

### 3.3 Structure of CED

As we have discussed, a CED is a well-formed XML document. Furthermore, we need to define the vocabulary of the tags used in CEDs. We shall give each of the tags used in CEDs a meaning so that any WebMC client is able to understand the CED. In this section, we will describe the structure of CED.
3.3.1 Reference and Collection

In WebMC, a CED is a structured collection of references. Each of the so-called reference refers to a specific document or resource in the user's PCE. For example, every personal document in a user's PCE should be referred by a reference in the user's CED. These references could be presented to the user as links on the web pages.

In WebMC, these references appear as Universal Resource Locators (URLs). Each of the references gives the URL this reference refers to. For example, a reference named "presentation slides" may have the URL of http://webmc.cs.ubc.ca/~naizhi/present.ppt. The WebMC middleware will be able to access this document via the URL.

This generalization gives CED better flexibility, as it no longer depends on any particular communication protocol to access PCE documents and resources. For example, a CED could be designed to use LDAP server to store many personal preferences. This architecture supports multiple mappings to the same data. For example, mailers and word processors all make use of author's contact book in an LDAP server, but they need different representation format for the same data. Users could set up two URLs pointing to the same data with different representation and they could be used by both applications.

Beside the URL, a reference also has some other properties. Every reference has a name, which is used to identify this specific reference. Every reference also has a mimetype field to denote the mimetype of the document this reference refer to. However, the meta information usually seen in a filesystem, such as length and modification time, are not in a reference and are revealed when the actual document get accessed using the URL.

The easiest way to store the URLs in a CED is to store them explicitly in every reference. In this case, whenever an application accesses a document, the URL
will be already given explicitly in the CED and the document will be accessed using the given URL.

However, to support better flexibility and simplicity, we do allow users to use alternative ways to construct the URL for each reference. For example, one construction method proposed is to concatenate the URL of the containing collection with the name of the reference. The details for the different methods to generate URLs will be discussed in the following sections.

These references reside in CED in a structured way based on the concept of collections. Recursively, a collection is a container that contains zero, one or more references or collections. In this way it resembles the directory structure in a file system.

Collection has a number of properties. First every collection has a name to identify itself. A collection also has a property called URL Info. Similar to the URL in a reference, this URL Info has the format of a URL; different from the URL in a reference, this URL Info is used to construct URLs for the references in this collection. A collection also has some Additional Operations, which will be discussed in the next section.

In a summary, references in a CED could be described to be loosely coupled, i.e. different protocols may be used to access different references in one CED. Different references may even be put on different servers on the Internet. In other words, the internal structure of a CED is separated from the storage structure of the PCE data.

### 3.3.2 Translation and Additional Operation

In order to support mapping of CED to a file system, we introduced translations and additional operations into references and collections. This section will tell why they are introduced and what they are.

In a file system, an application may look for a particular config file in a specific directory. For example, Adobe Acrobat Reader will store the configuration
settings in file $HOME/.acrorc (in UNIX systems). Suppose the reference to this configuration files is in collection PersonalPreference and called acro_resource. When a WebMC user invokes Acrobat Reader, this application will still go to the $HOME directory to look for the configuration file. The WebMC middleware should be aware that $HOME/.acrorc corresponds to the acro_resource in CED and access this file using the right URL in the CED.

Under this situation, a translation is necessary to give the right path name for a specific reference in CED. A translation is defined as the path name in a file system that a specific reference or collection is to be mapped to.

Since the collection structure resembles the directory structure in a file system, one natural way to map to a file system is to directly map a collection to a directory with the same name. This kind of translation means that the CED is structured according to how applications use them.

A URL only provides the location and access method of the resource. However, a document in a PCE may be updated or deleted. This could not be achieved with URL only. In WebMC, we introduce the concept of Additional Operation, which defines other operations that user can do on this reference or collection. For example, there could be an “update” operation on a reference, as well as a “create new document” operation on a collection. These additional operations should be used by the WebMC middleware to provide support to web-top applications.

The translation and additional operation of a specific reference could be inherited from the containing collection. For example, a translation of a reference could be the translation of its containing collection plus the name of the reference.

3.3.3 Presentation-Oriented Structuring of Collections

Collection in CED provides us the foundation to organize references. The next step is to setup the collections, and decide which collection each reference should go to. From the above discussion, we know that one simple solution is to organize
the references according to their position in the mapped file system, i.e. according to how they are used by applications.

In WebMC, a different approach is used to organize them according to how they are presented to the user or how the user sees them. In other words, the structure of CEDs is presentation oriented. For example, personal preferences are grouped together because they are usually not shown to the user directly.

The file system in a traditional operating system has long been deemed a good abstraction for data storage and access. The presentation-oriented structure of CED could be considered a level higher than the file system, which establishes itself through translations into a file system for data accesses and pursues data storage through Internet.

Figure 3.1 Structure of CED

![Diagram of CED structure](image)
The structure of CED is illustrated in Figure 3.1. The CED has two top-level collections: A and B. A contains another collection C and reference 1 – 3. Each of the references is presented to the user as one item on the web page. Collection A and reference 2 are mapped to specific directory “right” and file “son” on the mapped file system. Reference contains attributes like Name, Translation and URL. The URL of reference 2 points to a file on an HTTP server and will be downloaded to client when accessed.

3.3.4 Elements Overview

CED is the root element containing everything. It is the starting point of a CED. All references and collections in the CED are contained by a mother collection: RootCollection.

The RootCollection element contains three elements: ServicePool, DocumentPool and PreferencePool. These three elements are collections, containing Services, Documents and Preferences, respectively.

ServicePool contains zero, one or more “Service” elements. It can also contain ServicePool recursively. Service elements are references to services, i.e. web-top applications.

DocumentPool contains zero, one or more Document elements. It can also contain element DocumentPool recursively. Document elements are references to user’s personal documents. WebMC middleware uses URL in this reference to access the document.

PreferencePool contains zero, one or more Preference elements as well as PreferencePool. Preference elements are references to user’s personal preference documents for specific applications. WebMC middleware should enable applications to access these preference documents, possibly through a file system style support.

The structure of the elements is shown in Figure 3.2.
By using XML to store CED – the meta information of a user's PCE, it is possible to organize it in any ways we want. In WebMC we choose to organize CED it in a user's view rather than in a computer's view. This fits our requirement to be platform independent.

### 3.4 Element Details

This section gives the detailed description of each element used in CED. The functions of each element, their format in DTD grammar, and some descriptions are given for each of the elements. In appendix A, we will give a complete DTD for CED design.

![Figure 3.2 The Structure of Collections in CED. Ovals are collections and squares are references.](image)

#### 3.4.1 Element CED

The root element of a CED.

```xml
<!ELEMENT CED (UserInfo, RootCollection)>
<!ATTLIST CED Version CDATA #REQUIRED>
```
This is the wrapper element for a CED. It must have one attribute Version, giving the version of CED format used.

This element must contain a Userlnfo element giving information regarding the user him/herself. The second sub-element of this element is RootCollection, which is the root collection for all references.

### 3.4.2 Element UserInfo

This element describes the information related to the user. It is defined as

```xml
<!ELEMENT Userlnfo (Name, Description, Supplementarylnfo)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Description (#PCDATA)>
<!ELEMENT Supplementarylnfo (#PCDATA)>
```

This element gives the user-specific information of a CED. Element UserInfo includes three sub-elements: Name, Description and Supplementarylnfo. These three sub-elements describe the name of the user, the description of the CED and any other supplementary information that the user wants to add to this CED.

The information inside the UserInfo element may be displayed to the user as well. For example, the Name information could be put on the desktop to show whose CED is currently being displayed.

### 3.4.3 Element RootCollection

This element is the collection that contains all references and all other collections. It is defined as

```xml
<!ELEMENT RootCollection ( CollectionInfo, Self, (DocumentPool | ServicePool | PreferencePool )* ) >
<!ATTLIST RootCollection Name CDATA #REQUIRED>
```

This element has one mandatory attribute Name, which gives the name of the root collection. This element has two mandatory sub-elements: CollectionInfo,
which gives the collection-related information for this collection; and Self, which is a reference referring the CED XML document itself.

The other three kinds of optional elements, DocumentPool, ServicePool and PreferencePool are different collections under the root collections, which contains the three different types of references.

### 3.4.4 Element CollectionInfo

The common sub-element for all collection type elements.

```xml
<!ELEMENT CollectionInfo (URLInfo? , AdditionalOperation? , Translation? ) >
```

Element CollectionInfo is the common element to store information related to a collection. This element is contained by collection elements, i.e. ServicePool, DocumentPool and PreferencePool and is responsible for presenting generic collection information.

This element has three optional sub-elements: URLInfo contains the URL information that is used to generate the URL of contained references; AdditionalOperation defines the additional operations the WebMC middleware can do on this collection; Translation defines the mapping of this collection to the target directory in a file system, to be accessed by applications.

### 3.4.5 Element URLInfo

```xml
<!ELEMENT URLInfo ( #PCDATA ) >
```

This element represents the relevant URL in a collection. In a collection, it is used in conjunction with AdditionalOperation to fulfill tasks like create new collection or create new reference. The URL in the collection usually points to the storage where new elements could be placed.
3.4.6 Additional Operation Elements


This is a common element used by ReferenceInfo and CollectionInfo that defines available operations that could be done on this collection or reference.

The former two operations could only be done to a collection. When a collection or a reference needs to be created under this collection, or this collection is to be deleted from containing collection, the AdditionalOperation element is used to determine whether such an operation is valid, and how such an operation should be carried out.

The Update sub-element only applies to a reference. This element defines what action should be taken when a referenced file is to be updated.

<!ELEMENT CreateCollection ( NameAppendSimple | NameAppendInformServer )>
<!ELEMENT CreateReference (NameAppend | NameAppendSimple )>
<!ELEMENT Update ( DefaultUpdate | SpecialURLUpdate )
<!ELEMENT DeleteSelf ( SpecialURLDelete )>
<!ELEMENT NameAppendSimple EMPTY>
<!ELEMENT NameAppendInformServer EMPTY>
<!ATTLIST NameAppendInformServer URL CDATA #REQUIRED>
<!ELEMENT NameAppend EMPTY>
<!ELEMENT DefaultUpdate EMPTY>
<!ELEMENT SpecialURLUpdate EMPTY>
<!ATTLIST SpecialURLUpdate URL CDATA #REQUIRED>

These elements represent the additional operations that could be done on a specific reference or collection. For example, when doing a CreateReference operation on a collection, the NameAppend element

37
means that the URLInfo field of the created reference will be the URLInfo of this collection concatenated with the name of the created reference.

3.4.7 Element Translation

<!ELEMENT Translation ( #PCDATA )>

This element is a common element used by both CollectionInfo and ReferenceInfo element for translation into a file system path. The string in this element is the path name relative to the root point. The path name use the separator "/" and should be translated correctly onto different operating systems and file systems.

3.4.8 Element Self

<!ELEMENT Self ( ReferenceInfo )>
<!ATTLIST Self Name CDATA #REQUIRED>

This element is a reference giving the information of the CED itself. It has 3 sub-elements: URL gives the URL information as to where this CED is downloaded from; Translation is an optional element that gives the translation to a file system; AdditionalOperation gives information for the WebMC middleware to update or do other operations on the current CED.

3.4.9 Element ReferenceInfo

<!ELEMENT ReferenceInfo ( MimeType, URL? ,
AdditionalOperation? , Translation? ) >
<!ELEMENT MimeType ( #PCDATA )>
<!ELEMENT URL ( #PCDATA )>

This is a common element used by references to store the information to a reference. It has four sub-elements: MimeType is the mimetype of the referenced file; URL is the URL where the actual file is stored, represented by the string in this element; AdditionalOperation defines what operations could be done
on this element; Translation defines where this reference is to be put in a file
system.

3.4.10 Element DocumentPool

<!ELEMENT DocumentPool (CollectionInfo, (DocumentPool |
Document)* ) >
<!ATTLIST DocumentPool Name CDATA #REQUIRED>
<!ELEMENT Document ( ReferenceInfo )>
<!ATTLIST Document Name CDATA #REQUIRED>

This element is the collection for references referring to personal documents in
user's PCE. It contains a mandatory sub-element, CollectionInfo, for the
collection-related information. It can contain multiple elements of Document,
which is a reference to a personal document in the PCE, and/or multiple elements
of DocumentPool recursively.

3.4.11 Element ServicePool

<!ELEMENT ServicePool (CollectionInfo, (ServicePool |
Service)* ) >
<!ATTLIST ServicePool Name CDATA #REQUIRED>
<!ELEMENT Service ( ReferenceInfo )>
<!ATTLIST Service Name CDATA #REQUIRED>

This element is the collection for references referring to services in user's PCE. It
contains a mandatory sub-element, CollectionInfo, for the collection-related
information. It can contain multiple elements of Service, which is a reference
to a service in the PCE, and/or multiple elements of ServicePool recursively.

3.4.12 Element PreferencePool

<!ELEMENT PreferencePool (CollectionInfo,
(PreferencePool | Preference)* ) >
<!ATTLIST PreferencePool Name CDATA #REQUIRED>
This element is the collection for references referring to personal preferences in user’s PCE. It contains a mandatory sub-element, CollectionInfo, for the collection-related information. It can contain multiple elements of Preference, which is a reference to a personal preference file in the PCE, and/or multiple elements of PreferencePool recursively.

3.5 Summary

This chapter presents the design of the Computing Environment Descriptor (CED). CED is the central piece of a user’s computing environment to interact with WebMC clients. Every CED must be defined on top of a standard, machine-understandable vocabulary.

The basic requirements to the design of CED are to describe user’s PCE and allow any WebMC client to retrieve necessary meta information from it. There are also a number of other requirements that a CED must conform to.

XML is chosen to be the fundamental language of CED. Based on the structure of familiar markup language yet providing capability to define vocabulary to be understood by programs, XML gives us many benefits over other languages.

The structure of CED is based on the concept of reference and collection. Based on these two concepts, the structure of a CED follows the way it is presented to users rather than it is used by applications, or so-called presentation oriented.

Finally the elements to construct a CED are defined in detail.
Chapter 4

Prototype Design and Implementation

To demonstrate the WebMC computing model, we designed and implemented a WebMC prototype. This chapter presents the design and implementation of this WebMC prototype, which is a result of the discussion of the WebMC computing model in the previous chapter. This chapter first presents the background and requirements of the prototype design.

This chapter then discusses the architecture of the WebMC prototype, and after that various components of the prototype is discussed in detail. Finally a summary is given about the pit and falls of the prototype design.

4.1 Prototype Background and Requirements

The goal of the WebMC prototype is to demonstrate that the concept of WebMC is feasible, useful and superior over the traditional systems. It is not meant to provide full support of user mobility. Thus, some aspects of the WebMC prototype may not provide all functionalities discussed in the previous chapter. For example, security and trusting are not considered in this prototype.

As mentioned in previous chapters, the two kinds of services is the basic functionality to support. This WebMC prototype must support web-top applications’ accesses to user’s PCE file and assume local registration function.
Besides these two basic requirements, we outlined several other principles that we should follow during the prototype development. They are explained in the following paragraphs.

The prototype is to support user mobility, and besides that there are certain limitations and requirements that we should follow in the WebMC prototype design. The first one is the project size. *The WebMC prototype project should fit in a master's thesis.* There are a number of design choices that are more or less impacted by this limitation. For example, universal plug-in was selected over modifying the Mozilla source code partly because of this constraint. At the same time, the prototype implementation shouldn’t require expensive devices, software licenses, or source codes.

According to the challenges discussed in the first chapter, a user’s personal computing environment must be cross-platform. A specific WebMC implementation may be platform dependent, but the user’s computing environment must be in a platform-independent form. *The WebMC prototype must support a platform-independent personal computing environment, while the prototype itself may be built on platform-dependent components.* This prototype should be able to demonstrate that it is feasible to associate a platform-independent representation of a user’s personal computing environment with platform-dependent elements.

Another design requirement is to maintain backward compatibility. *The WebMC prototype should be able to run existing applications and provide mobile computing support with little or no modification to the applications themselves.* Backward compatibility has been proved very important in many fields, and it’s no exception in this case. Considering the large amount of current available applications, it’s very desirable that a new system with WebMC support will still be able to run most applications that users are familiar with. This prototype will try to demonstrate that such a system is viable.
4.2 Prototype Platform

The WebMC prototype is built on top of Suse 6.3 Linux Operating System. This system is of kernel version 2.2.13. The web browser in the middleware comes from the Mozilla Milestone 15 [Mozilla2000].

We pick up Linux mainly for two reasons: 1) Linux is becoming more and more popular now. Many software applications, including some task-critic applications, are developed on top of Linux. Thus the prototype would verify that building WebMC on such a system is viable. 2) Linux is an open source software. We anticipate possible extensions to the Linux kernel system, thus open source means a lot to us.

With the operating system decided, the benefit of using Mozilla over the other major browser on the market is obvious. Not only it is designed to be cross-platform, it has good extensibility through its plug-in interface and the modular design. The open source of Mozilla also gives us greater flexibility in design and implementation.

4.3 Implementation Background

There are two technologies that this prototype implementation relies on: the Linux VFS technology and the Mozilla. This section provides a brief overview of the two technologies, in the hope to help readers to better understand the rest of this chapter.

4.3.1 Linux Virtual File System

Among the good features of variants of UNIX operating systems is being able to support different kinds of file system simultaneously. This is done through the Virtual File System, or VFS. Illustrated in Figure 4.1, the VFS layer is a framework providing a link between the file I/O system calls and a clearly defined interface with which individual file systems can be implemented.
In Linux, the most important concept is inode, which represents a single entity (i.e. a file or a directory) in VFS. The inode structure contains two parts: the file system independent part that is used by VFS, and the file system dependent part that is used by the underlying file system. The first part contains information like file mode (whether it is a directory, link or a regular file), modification time and etc. The later part is implemented as a union of file system dependent structures.

Different file systems register its operations using two function tables (array of function pointers): the inode operation function table and the file operation function table. The file I/O system calls are passed to the VFS layer, which picks up the correct function table by file system mount points and calls the appropriate function in the table. A file system does not need to implement all functions; the Linux VFS uses default operations for unimplemented functions.

4.3.2 The Mozilla and Plug-in Architecture

Mozilla is designed using a modularization technique called XPCOM (Cross-Platform Component Object Model). This modularization gives us a lot of convenience in that it is a lot easier for a plug-in to get the service in other parts of Mozilla.

Mozilla is based on modules. Each module exposes its interfaces to other modules via pure virtual classes containing only pure virtual functions. All interfaces inherit from a single nsISupports interface, which defines reference counting and

![Linux Virtual File System](image)
interface casting functions. A plug-in also obeys the rules of XPCOM and uses resources inside Mozilla via XPCOM interfaces.

4.4 Prototype Architecture

The basic two goals of the WebMC middleware is to provide two kinds of services to web-top applications, i.e. PCE document access and local resource registration. In the prototype, a new file system is introduced to support access to PCE document, while an application description table is used to register and manage local applications. In this section, we will discuss the architecture of the middleware prototype.

4.4.1 PCE File Access Service Provided via XIFS

The design of CED has enabled the mapping of PCE files to a file system. This prototype makes use of such mapping to build a file system plugged into the Linux Virtual File System, to provide web-top applications accesses to user’s PCE documents. We call this file system XIFS, which stands for XML Internet File System. It uses information in the CED, which is written in XML and uses the vocabulary defined in chapter 3, to store the meta information of the file system.

Such a file system style support has a number of benefits. A lot of web-top applications uses file system interface to access preference files and files to be processed. Thus building such a file system could provide user mobility to these applications. Modern operating system including UNIX, Linux and Windows support mapping of file systems to specific data source.

Since such a file system mapping is transparent to applications, it satisfies one of the requirements mentioned before: to provide user mobility service to existing applications with little or no modifications to these applications.

In our prototype, the XIFS is a Virtual File System mounted on the home directory of the current user. Because most applications on Linux put their
preference documents under the home directory, the applications will access the preference files just like they access files in local disks.

Presumably, it will be convenient for WebMC middlewares on all UNIX platforms to mount on home directories. Nevertheless, for other platforms that don’t have the concept of home directory, it may be favorable to organize the mapped file system in other ways. For example, Windows 95 series doesn’t have the concept of home directory and the preference files of applications are spread everywhere in the file system.

XIFS uses information in CED to construct the file system. The information in CED becomes the meta information of XIFS. Since CED is written in XML, so XIFS could be considered using XML to describe the meta info. This is also why it is called Xml Internet File system.

**4.4.2 Providing Application Registration Service via Universal Plug-in and Application Description Table**

This prototype provides service for local resource registration through an application description table. Considering the limit on the project size, the simple way of using such a table was chosen over more general, flexible approaches.

In this prototype, web-top applications in the format of a Mozilla plug-in are automatically registered with the Mozilla web browser, thus automatically registered with the whole middleware. A plug-in is invoked when Mozilla comes across a mimetype registered by the plug-in.

Stand-along applications are not known to Mozilla. This prototype support stand-alone applications via a *Universal Plug-in*, which registers all the mimetypes that the stand-alone applications can handle. When Mozilla encounters such a mimetype, it will activate the universal plug-in, which in turn invokes the corresponding application.

The stand-along applications make themselves known to the universal plug-in by inserting an entry in a file called Application Description Table. This text file
contains the information like invoking path, supported mimetype and if it supports file name as a argument.

This prototype does not expose the CED structure and elements to web-top applications. However, a full local resource registration service should provide API for web-top applications to access the CED. These applications will then be able to be installed in user's CED automatically.

4.4.3 Components Overview

This WebMC prototype middleware contains a set of components to provide the two basic services mentioned before. In the following paragraphs, we outline these components, discussing their functional responsibilities, APIs, inter-component communications and more. We will introduce the different components in the prototype middleware, and then carry out the discussion via a scenario that walks through the functions of these components.

As illustrated in Figure 4.2, the prototype middleware is composed of the mozilla web browser, the universal plug-in and the XIFS Redirector. The universal plug-in is loaded into the address space of the web browser when the web browser starts, and get activated when user starts a WebMC session. The universal plug-in contains two modules: the XIFS processor and the application handler.

The XIFS processor is mainly responsible for helping web-top applications accessing user's PCE files. It relies on Mozilla to communicate with remote server using various protocols. The application handler is responsible for invoking the web-top applications.
A user first logs in by giving the URL of his CED to the WebMC middleware. He or she types in the URL into the web browser’s “Address” field, and the web browser will download the CED file with proper authentication. The CED is presented to the user with a pre-selected style. This style is written in XSL language, which retrieves the content of CED and present the content to a user. The user is happy that he can see the same desktop as his/her office computer, since the same CED and style is used in both situations.

The loading of the CED also triggered the universal plug-in to be active. The CED is internally represented as DOM (Data Object Model) elements and used by the XIFS processor module inside the universal plug-in.

From the web page shown on the screen, the user selects a link representing a specific type of service s/he wants to use. This request is interpreted by the web-browser to invoke the universal plug-in. The application handler module in the universal plug-in will look into the application description table and invoke the corresponding application to process this request.
This application starts and accesses the user’s personal preference files. The requests to access the files are intercepted by the XIFS redirector and re-directed to the XIFS processor in the universal plug-in via our asynchronous channel. In our prototype, these preference files must reside under the $HOME directory or its sub-directories to be intercepted by the XIFS redirector, because it is a VFS mounted on the $HOME point.

After it gets the requests, the XIFS processor will look in the user’s CED to find the corresponding reference for the requested file. Failing to find the reference means that the file does not exist under the file system. If it successfully finds the reference, the URL information in the reference will be retrieved and analyzed. The XIFS processor will interact with the Mozilla web browser to retrieve the information and pass the information back to the application via the XIFS redirector.

After the user finishes with his/her session, s/he logs out from the system and the CED will be written back to the server if necessary.

The XIFS processor, application handler and the XIFS redirector modules together with the Mozilla web browser provide the two basic services necessary for a WebMC middleware. The application handler and the XIFS processor are put together in the universal plug-in because they both interact with Mozilla.

4.5 XIFS Redirector

The XIFS redirector acts as the prototype’s VFS component to provide transparent access to user’s PCE files for web-top applications. Web-top applications’ requests to access user’s PCE files are intercepted by this component and redirected via a special channel to the XIFS program, which runs as a plug-in of mozilla web browser.

XIFS is a new file system type registered with Linux’s VFS (Virtual File System). The $HOME directory is used as the mount point to mount this file system. All file operations under that directory will be processed by XIFS redirector code.
This VFS structure provides the possibility for us to support transparent access to user’s PCE files.

The XIFS redirector is a dynamic loadable kernel module. It could be inserted into and removed from kernel address space at any time. A user just has to run one command to start the program like how other user level program is run. For the designer, it is also much easier because s/he doesn’t have to link with kernel and reboot machine many times.

The principle of designing XIFS is “the simpler the better”. The complexity is left at the most possibility to the XIFS processor, which runs in the user level. The requests from the application are still quite “raw” when they get redirected to the XIFS processor. The XIFS redirector, on the other hand, is kept simple and therefore more robust.

The communication channel between the XIFS program and the XIFS redirector is based on two special files, webmc_in_file and webmc_out_file. These are two regular files residing on the XIFS file system. However, they are not treated as the regular files: they serve as the bridge between the XIFS processor and the XIFS redirector. A read operation on the webmc_in_file returns the file operations by a web-top application; and a matching write operation on the webmc_out_file gives the result back to the web-top application. Special data structures are used to read and write on these two files. These structures are the protocol between the XIFS processor and the XIFS redirector.

Figure 4.3 shows the information flow between web-top applications and the XIFS processor. Thick arrows are file operations on normal files and the special files. Upon initialization, the XIFS processor will open the webmc_in_file in exclusive read-only mode and open the webmc_out_file in exclusive write-only mode. It will then go into an infinite loop to read from the webmc_in_file. This read operation will return the file operation requests from the web-top applications if there exists any, or hangs until one comes. The XIFS processor writes the result of a specific request to the webmc_out_file and this result will
then be re-directed to the webmc_out_file. This write operation returns immediately.

The protocol between the XIFS redirector and XIFS processor is based on request/reply packets. The XIFS processor reads request packets from webmc_in_file and writes reply packets to webmc_out_file. Each request or reply packet has two parts. The first part is a header that contains a sequence number, a request or reply type, a request or reply value and the size of the following data. Following a request or reply packet is the variable length operation-specific data. This data is passed through the channel for interpretation by operation routines.

```
struct reqpak {
    unsigned long seq; /* sequence to match req/rep */
    unsigned int rtype; /* request type */
    unsigned long rval; /* request value/return value */
    unsigned long size; /* size of body */
};
```

Figure 4.4 Header Structure of Protocol between XIFS Redirector and XIFS Processor
The header structure is shown in Figure 4.4.

Because the XIFS redirector is designed to be simple, so almost all the VFS operations are passed to the XIFS processor, including those inode operations. The operations are shown in Table 4.1. Each of the operations has its specific data area following the request header.

<table>
<thead>
<tr>
<th>Operations</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELETE_INODE</td>
<td>Delete a file or a directory</td>
</tr>
<tr>
<td>INODE_CREATE</td>
<td>Create a file or a directory</td>
</tr>
<tr>
<td>INODE_LOOKUP</td>
<td>Lookup file or directory properties</td>
</tr>
<tr>
<td>FILE_READ</td>
<td>Read content from a file</td>
</tr>
<tr>
<td>FILE_WRITE</td>
<td>Write content to a file</td>
</tr>
<tr>
<td>FILE_OPEN</td>
<td>Open a file</td>
</tr>
<tr>
<td>FILE_CLOSE</td>
<td>Close a file</td>
</tr>
<tr>
<td>FILE_REaddir</td>
<td>Read directory entries from a directory</td>
</tr>
</tbody>
</table>

Table 4.1 XIFS Redirector Operations

However, it is the XIFS processor’s job to interpret these operations in a proper manner. Some operations, such as INODE_LOOKUP, could not be completed in exactly the same way as a local file system.

4.6 XIFS Processor

The XIFS processor is the central module to provide web-top applications with transparent accesses to user’s PCE files. The XIFS processor gets information from the CED, interacts with Mozilla to retrieve contents from Internet, and communicates with XIFS redirector to get the requests and pass the results. In the following part of this section we discuss in detail how the XIFS processor achieves this goal.
Constructing the file system from CED

The XIFS processor makes use of CED, which stores the meta information of user's PCE files. As we have discussed before, a CED is a collection of references to URLs. The XIFS processor must be able to construct a file system structure from the information in the CED.

The CED design incorporates considerations for constructing a file system to provide system service to web-top applications. Specifically, one special element "Translation" in the references and collections stores the information as to where the reference or collection is to be placed in the file system. This translation, however, is one way only, i.e. there is no explicit translation from file system back to the CED structure.

To respond to the file access requests from web-top applications, the XIFS processor constructs an internal tree to represent the target file system. Each node represents a reference or collection that is to be translated into a file or directory in the XIFS. This internal tree structure contains all meta information recovered from the CED, but it does not contain any actual content of a specific file. With this internal tree, the XIFS could easily go into the tree and find the reference that corresponds to the requested file.

In order to construct such a tree, all elements of the CED is to be parsed when the user logs in and his/her CED is downloaded. Each collection is to be translated into a node of directory type; and every reference is to be translated into a node of regular file type. When parsing a specific reference or collection, the XIFS processor looks into the "Translation" sub-element, and create the corresponding node, fill in the correct content in the node, and put the node in the right place in the tree.

Each of the nodes in the tree corresponds to an inode in the XIFS redirector. A unique number is allocated to every node and the processor uses this number to fill in the inode number field in the packets passed to the redirector.
The structure of a node of the tree is shown in Figure 4.5.

Because Linux VFS uses inode number to uniquely denote a file or directory, we need some ways to get the tree node from the inode number quickly. In XIFS processor there is a hash table from inode numbers to the nodes in the tree. The XIFS processor can easily get a pointer to the node structure when given an inode number from XIFS redirector.

**Getting requests and processing events.**

During initialization, the XIFS processor tries to open the two special files, `webmc_in_file` and `webmc_out_file` under `$HOME` directory. The middleware process will abort if this operation fails. It then spawns two threads: the main thread for event processing, and the listening thread for receiving requests from the XIFS redirector.

The listening thread repeatedly calls `read()` system call on the `webmc_in_file`. This system call will return with a request from the web-top applications. If there isn’t any request available in the queue, the system call will hang until a web-top application accesses a PCE file, causing the XIFS redirector to generate a request to pass to the XIFS processor.
Struct t_node

/* the following fields come from the inode of VFS */
unsigned int ino; /* inode number */
unsigned short mode; /* dir or regular file */
nlirik_t nlink; /* number of directories containing this node */

uid_t uid; /* user id, to be set to current user */
gid_t gid; /* group id */
off_t size; /* size of file or dir */
time_t atime; /* access time */
time_t mtime; /* modification time */
time_t ctime; /* creation time */

/* the following fields are from the CED or for internal use */
CedElement * element; /* the element in CED */
unsigned short status; /* bit string of the internal status */

struct t_node * parent; /* containing dir */
struct t_node * fson; /* if dir, first contained file/dir */
struct t_node * obrother, ybrother; /* file/dir in the same dir */
nsIPluginStreamListener *nlistener; /* stream listener, if any */
nsICachedNetData *pcache; /* cached entry, if any */

Figure 4.5 Data Structure of The Nodes in The Internal Tree

Whenever a request is read from the webmc_in_file, the listening thread will generate an event containing the request information to post on the main event thread, where the actual processing happens. The result of the processed request will be written back to the XIFS redirector via a write() system call on the webmc_out_file.

The main event thread is where most of the things get done. This thread implements an event loop. The events and event loop use the event facility provided by Mozilla.
The event scheme described here is asynchronous, meaning that a request may be picked up and processed even if the previous request has not been finished. This feature is essential to XIFS because most of the requests have some Internet communication thus cost longer time.

**Networking and caching with Mozilla**

The XIFS processor relies on Mozilla to carry out network and cache operations. Network operations may be necessary when a web-top application accesses a PCE file and the request is redirected to the XIFS processor.

Mozilla provides us a set of network libraries that a plug-in can use to retrieve data from Internet servers. The XIFS processor uses these libraries to communicate with Internet. Because Mozilla network libraries are able to handle a number of different protocols, the XIFS processor does not contain any code relating to protocol-specific communications.

The two key API functions are GetURL() and PostURL(). A parameter for both function calls is char * URL, which the XIFS processor should construct from the CED fields. These functions will take all protocols that Mozilla supports, thus there is no need for XIFS processor to understand the protocol in a particular URL.

The Mozilla also has some libraries to facilitate caching of network data. The XIFS processor makes use of these libraries to store the content of a specific URL. The XIFS processor also relies on Mozilla to handle all cache replacement.

**Interpretation of operations**

**The following is the operations on a reference node:**

**INODE_LOOKUP** If the node has never been downloaded, start accessing the referenced URL. This routine will return after the header of the response is available. The fields in the result packet to XIFS redirector (such as file length) are filled with the response from the remote server.

If the node has been downloaded before, return the fields in
t_node. This status is stored in the cstatus field.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE_OPEN</td>
<td>Checks if the file is to be opened for read or for write. If it is to be opened for write, then we should check if element -&gt; AdditionalOperation contains update capability.</td>
</tr>
<tr>
<td></td>
<td>If the URL exists in cache, then add a reference to the cache entry, to avoid this cache entry being swapped out.</td>
</tr>
<tr>
<td></td>
<td>If the URL does not exist in cache, then start accessing the referenced URL. This routine will return after the header of the response is available.</td>
</tr>
<tr>
<td></td>
<td>The fields in the t_node are updated with the new fields. Add a reference to the cache entry too.</td>
</tr>
<tr>
<td>FILE_READ</td>
<td>If the file is current being downloaded (nsIPluginStreamListener != NULL) and the file pointer &gt;= current offset, read from the current nsIPluginStreamListener.</td>
</tr>
<tr>
<td></td>
<td>Under other situations, read from the cache entry.</td>
</tr>
<tr>
<td>FILE_WRITE</td>
<td>Write to the cache entry.</td>
</tr>
<tr>
<td>FILE_CLOSE</td>
<td>De-reference the cache entry. If the file has been updated, construct the Post URL from the CED, and Post the URL to the remote server. This operation returns after the data has been successfully posted to the remote server.</td>
</tr>
<tr>
<td>INODE_DELETE</td>
<td>Checks the additional operation field in the CED. If the operation is permitted, then construct the URL and Post the URL to the remote server.</td>
</tr>
<tr>
<td></td>
<td>The element is removed from the CED and the CED is marked “updated”.</td>
</tr>
</tbody>
</table>

The following is the operations on collection nodes:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INODE_LOOKUP</td>
<td>Return the fields in the t_node structure. The most important field that should be filled in is the mode field, in which the</td>
</tr>
</tbody>
</table>
XIFS processor should set the "directory" bit.

FILE_OPEN      Do nothing.
FILE_READDIR   Return the dirents that is contained under this collection.
FILE_CLOSE     Do nothing.
INODE_CREATE   If the requested object is a regular file and the additional operation “CreateNewReference” is defined, create the new reference according to the policy defined in the additional operation and insert an XML element into CED.

If the requested object is a directory and the additional operation “CreateNewCollection” is defined, create the new collection according to the policy defined in the additional operation and insert an XML element into CED.

4.7 The Application Handler

The application handler is responsible to register the mimetypes for stand-alone applications and invoke them when necessary.

The application description table is a text file, in which each a line represents an application. Each line has several columns: application name, application invoking path, and mimetypes that this application supports.

Upon initialization, the application handler will register all the mimetypes in the application description table. After that, when Mozilla meets any of the registered mimetypes, it activates the application handler by calling a specific function in it. In the function the corresponding application is invoked with proper parameter.

4.8 Summary

This chapter presents the design and implementation of the prototype middleware. The goal of this prototype is to prove the feasibility and usability of the WebMC
system. This prototype is implemented on top of Linux and using Mozilla source code.

The basic requirement on this prototype is to support the two services to web-top applications. A number of other requirements and constraints are also outlined.

A file system is developed and inserted into the Linux VFS system and a universal plug-in is developed to work with Mozilla. These modules communicate with each other to provide the services to web-top applications.

After that, the data structures, basic operations are also discussed in detail.
Chapter 5

Related works

There are a number of research projects done in recent years that are related to mobile computing or user mobility. These projects provided different solutions to different problems, and more importantly, these projects have defined the area of mobile computing and user mobility.

In this chapter, we will discuss the research projects on mobile computing conducted in universities all over the world. First we will give an overview on terminal mobility support projects, and two projects are given as example research in this area. Then we will introduce several projects focusing on supporting user mobility, including UPC, NetChaser and Jini. A comparison is given in each section.

5.1 Terminal Mobility Systems

There have been a lot of researches done on the topic of supporting terminal mobility, i.e. supporting a specific device to move from place to place. The mobility of users and their computers has put mobile resource constraints such as limited wireless bandwidth, limited battery life and possibly changing environment. These research projects examined the impacts of these constraints
on information services and on the applications and tried to provide solutions to solve the impacts.

Some of the projects focus on adaptation of mobility, which covers various strategies and techniques in how systems and applications respond to the environmental changes and the resource requirements. The adaptation techniques were divided into categories shown in Figure 6.1 in [Satya1996]. At the left extreme, adaptation is entirely the responsibility of individual applications with no system support, while the right extreme places the adaptation responsibility solely on the system and requires no changes to applications. Between them is a spectrum where applications cooperate with system for adaptation.

Some research projects choose to put certain operations normally performed on clients to be performed on resource-rich servers to get around the resource limitations on the client. For example, [Joseph1996] employed mobile object concept to dynamically transmit an object (code and data) between client and server, to reduce client-server communication requirements. Some other projects adopted application-specific proxies which act as intermediary between clients and servers to perform computation-intensive tasks on behalf of clients.

**Comparison with WebMC:** The focus of the terminal mobility projects is different from WebMC. WebMC tries to give an approach for high-level integration of

![Figure 6.1 Range of Adaptation Strategies](image-url)
mobile computing and user mobility, while these projects tried to give solutions to the changing and limited resources such as network bandwidth. In the prototype, the middleware provides transparent access to user’s PCE files through XIFS, thus it could be considered to be application-transparent adaptation.

In the rest of this section, we will present two special cases on terminal mobility support: Odyssey and Rover, both attempting to solve the resource limitation on mobile devices.

5.1.1 Odyssey

Odyssey [Nobel1998] is a project conducted at Carnegie-Mellon University on application-aware adaptive mobile systems. The goal of Odyssey is to provide a platform to coordinate concurrent mobile applications for their diverse adaptive behaviors. It claims that diversity and concurrency in mobile computing calls for application-aware adaptation with centralized resource control.

To address the diversity of mobile applications, Odyssey categorizes data into different types. Each typed data will be treated with a specific adaptation policy. The system monitors resource usage of all types and enforce resource allocation decisions among these types. Each type of data is accessed through a specialized component called Wardens. A warden provides the support for the application to manage a data type effectively. The wardens communicate with a type-independent component called Viceroy, which serves as a centralized resource management facility. The system-level support is provided through an Odyssey VFS implemented inside NetBSD kernel.

There exists several kind of interaction between the application and the Odyssey system. First, applications communicate resource expectations to the Odyssey system. The system decides if sufficient resources are available to this application and tells it whether the requested resources are allocated. Second, when the availability of resources has changed, the viceroy notifies corresponding applications and the application is expected to respond to the notification by
adjusting its fidelity according to its own policy. Last, the application can have type-specific operations on the wardens of that type.

In the prototype Odyssey system, several applications were developed to demonstrate their ideas. A video player is developed based on public code and the corresponding warden is written for it. Fidelity is divided into 3 levels: JPEG compressed color frames at quality 99 and 50, and black-and-white frames. This video player will automatically adjust its fidelity according the availability of resources. Netscape is set to redirect its request via a warden, and then have different levels of fidelity on the images.

5.1.2 Rover

Rover is a research project carried out at MIT. The purpose of Rover is to provide a platform supporting application-aware adaptation for mobile client applications. In the same time, application-transparent adaptation could be achieved by developing Rover proxies for the original applications running on the mobile client.

Similar to DCOM and CORBA, a Rover application is required to be developed as a collection of Relocatable Dynamic Objects (RDOs). A RDO is an object (code and data) with well-defined interface. It is dynamic because it can be loaded from a server computer to a client computer, or vice versa, all in run-time. RDOs enable applications to migrate functionality dynamically to either side of a slow connection to minimize communication across the network.

Queued Remote Procedure Call (QRPC) is used to enable applications to continue to make non-blocking RPC calls even when the mobile client is disconnected, in which case requests and responses are exchanged upon reconnection. It is used to lazily fetch RDOs from the server to the client and the local copy of RDO is cached. If this locally cached RDO is modified by the application, the application need to use QRPC again to commit the updates to the server.
The interactions between Rover and its applications have four primary functions: create session, import, export and invoke. Applications use import to import a RDO to the client side. Once a RDO is imported, the application can invoke methods in the RDO to operate on it. Applications propagate each local update to a RDO back to the servers by issuing an export operation. The export procedure works in an operation-based manner, i.e. only the actual operation is propagated back to the server and re-executed in the server side.

An example application for Rover is the Rover Exmh – a Tcl/Tk based mail reader. It employs three levels of RDOs: mail messages, mail folders and lists of mail folders. These three levels of RDOs allow many user requests to be processed locally without inducing any network traffic. Some computation is migrated to servers. An example is doing a glimpse search of mail folders locally at the client, in which case the client can construct a search request RDO and send it to the server.

5.2 Universal Personal Computing

Universal Personal Computing is a project carried out at University of British Columbia. It could be considered the predecessor of this thesis. The goal of UPC is the same as WebMC, i.e. achieving user mobility, enabling mobile users to access user-defined Personal Computing Environment from any currently

![Figure 6.2 Architecture of UPC](image-url)
available network terminal, be it fixed or mobile and irrespective to geographical locations.

A prototype of UPC is implemented with Java and Corba [Zhu1999]. As its architecture is shown in Figure 6.2, it is constituted with two parts: home network and foreign terminal. Users should be able to access his own Personal Computing Environment defined in his home network from any foreign terminal.

When a user logs into a foreign terminal, the Initial Agent asks the Terminal Agent to locate the User Agent for this user, and obtain the set of home services (pre-defined in PCE). This set of home services is compared with the terminal constraints and decide if the services are also available locally, and giving the user a choice of using them locally if available.

Two kinds of services will be displayed to the user: i) those available at home and capable of being supported by the foreign terminal; ii) those available at the foreign network locally. When the user picks up a service from the user-interface, the UPC system will bring in the PCE containing the user preference of the application required, and then run the application (download first if necessary).

However, this prototype still suffer from the following problems:

(1) There is no distinct service provided by UPC to the applications.

(2) Files are not included in PCE. Users cannot access personal files that are part of the computing environment.

(3) Any modifications to personal preferences cannot be saved back to home server.

(4) Each terminal is supposed to have a local file describing certain properties of the terminal. This file is recognized only by the UPC system and composed manually by system administrator using a tool provided by UPC.

(5) Users have to learn how to use the specific interfaces of UPC system, which makes it less widely accepted.
(6) User interfaces of UPC are not customizable; every user uses the same one.

Comparison with WebMC: UPC could be considered the predecessor of WebMC project. They share the same goal: to support user mobility. However, UPC uses Java programming language to describe a user’s computing environment, while WebMC uses a web-based language XML to describe a user’s computing environment. This XML interface is presentation-oriented. In the mean time, UPC program is implemented in Java while WebMC middleware can be implemented in local binary. WebMC also defines distinct services that a WebMC middleware should support to web-top applications.

5.3 Agent-based Systems

NetChaser [Stefano2000] is an agent-based infrastructure for supporting user-mobility (as called personal mobility in [Stafano2000]) in accessing Internet information services. It’s carried out at University of Catania, Italy.

The goals of NetChaser and WebMC are somewhat similar to WebMC: to provide an infrastructure for supporting user mobility. NetChaser goes through the agent solution: mobile agents serve as a wrapper layer between the applications and the Internet, providing assistance to the users by following the user when they change working terminals. The requirements of NetChaser are composed of three key parts: adaptability, transparency and heterogeneity.

NetChaser requires multiple servers to handle working files and user profile settings. File servers enable users to access working files from host stations, and user servers are profile repositories and perform user authentication. A user profile includes web bookmarks, welcome web page, preferred download applications and addresses of the relative application servers, e-mail settings and so on.

Agents, which are also called assistants in NetChaser, reside on the client machine statically or dynamically to assist users. There are at least three kind of static assistants on each client machine: user assistants, user profile managers and file
server managers. A different service assistant agent exists for each kind of service [Stefano2000].

An example is HTTP assistant. In web browsing, the browser passes the HTTP requests first to the user assistant, which in turn passes the requests to the HTTP assistant. The primary function of HTTP assistant is to maintain HTTP session status like cookies, recently accessed URLs (Back and Forward buttons). Thus when a user moves from a host to another, s/he can get the same status back.

Comparison with WebMC: Sharing the same goal of supporting user mobility (called personal mobility in [Stefano2000]), NetChaser uses an agent approach. Each service (or application) has a specific agent to help the application access the application’s preferences. Another agent is used to help application access the user’s personal documents. However, the interface between the agents and the applications are not discussed. NetChaser approach is somewhat on the Laissez-faire side on Figure 6.1, i.e. each application handles its mobile characteristic itself. In contrast, WebMC uses a file system interface between the web-top applications and the middleware. Similar to WebMC, [Stefano2000] also mentioned that web pages are used to serve as user’s desktop, but the details are not given.

5.4 Resource Discovery Systems

Jini [Jini2000] is a network middleware developed at Sun Microsystems. The ultimate goal of Jini is to enable users and computational clients to access and manage hardware and software resources throughout the network in a flexible, easy and scalable manner. To the mobile world, it also provides users easy access to resources anywhere on the network while allowing the network location of the user to change [JiniWhy2000].

Jini is based on Java environment and makes use of RMI technology. All applications running with Jini could only be in the form of Java bytecode.
The most important abstraction of Jini is that of a service. A service is an entity (Java bytecode and data) that can be used by a person, a program or another service. A service provides certain software or hardware conveniences such as computation, storage, or a software filter. The purpose of Jini is to federate these services into a single dynamic distributed system for simplicity of access, ease of administration and support for sharing. Services are accessed via a service protocol, which is a set of interfaces written in Java programming language.

Jini provides an infrastructure for services to register themselves into the system (join) and for users to lookup registered services. A service registers itself with Jini system by providing a service object in Java bytecode to Jini system. The service object implements the interfaces for this service, i.e. those that the client uses to execute the service.

A client locates an appropriate service by its type along with descriptive attributes that are used in a user interface for the lookup service. Jini supports transaction, which encapsulate a series of operations to implement two-phase commit. Jini also supports distributed event interfaces to be used by Jini clients and service providers.

Comparison with WebMC: A Jini system provides the infrastructure for clients (users) and service providers (applications, hardware devices) to communicate without prior knowledge of the other party. The interface between these two parties, provided by the Jini Lookup Service, is a Java interface that could be registered with Jini and looked up by Jini clients. In WebMC, this interface is done through mimetype. Both of them need a standard for users to move from place to place. Jini only supports Java bytecode applications, while WebMC aims to support platform dependent binary code through the platform-dependent middleware implementation. Jini could be considered at a lower level of WebMC and be utilized for resource discovery part of a WebMC middleware.
Chapter 6

Conclusion and Future work

User mobility is another degree of mobility that is less pursued in the area of mobile computing. However, because of the inherent constraints on a mobile device and other factors, user mobility is necessary under many situations where pure terminal mobility may not suffice users' needs.

This thesis proposes a new computing model that tries to provide user mobility service to all applications through a system-level solution. This solution utilizes a platform-independent interface to fit a user's personal computing environment on a platform-dependent middleware that provides two services to web-top applications. A prototype middleware has been built to prove the feasibility of WebMC. This prototype uses a file system to support web-top applications access user's PCE files.

In the previous chapters, we have presented a complete image of WebMC systems. In this chapter, we shall now conclude by giving a summary of this thesis, presenting possible future works, and providing a final conclusion remark.
6.1 Summary

There has been a proliferation of research on terminal mobility problems. Terminal mobility assumes the fixed association of a user and a certain mobile device. However, this association may not always exist because of the inherent constraints on a mobile device and many other factors. Under these conditions, user mobility – another degree of mobility in which users could access his/her own computing environment from any Internet connected terminals – is necessary.

A system to support user mobility faces several challenges. First, a user’s personal computing environment must fit in a heterogeneous environment. Second, local resources on a WebMC client must be integrated into the user’s computing environment to best support user. There are also security issues and adaptation issues. However, these two issues are not addressed in this thesis.

WebMC Computing Model is proposed to be a system-level solution to address these challenges. In WebMC, a user’s personal computing environment is platform-independent and described by an XML document called Computing Environment Descriptor (CED). On different platforms, a WebMC middleware based on a web browser reads this document and present it to the user in exactly the same way as his/her home environment.

WebMC proposes a design of the vocabulary of CED. This design uses URLs to represent user’s PCE files and defines the mapping to a file system. The structure of CED is presentation-oriented, meaning that it is close to what user sees rather than how they are stored or organized physically. With this standard vocabulary, a WebMC middleware would be able to understand the CED and use it to support web-top applications access user’s PCE files.

To demonstrate the feasibility and usability of WebMC, a prototype is built on top of Linux and Mozilla. This prototype maps a user’s CED to the $HOME directory. This prototype also uses an application description table to register
local applications. An experiment is carried out in WebMC and two example applications are demonstrated.

6.2 Future work

WebMC model opens new opportunities for future work. The WebMC system, including the prototype could be improved in many aspects even within the definition in the thesis. Beyond that, there are also many new directions that could be developed and integrated into WebMC systems.

6.2.1 System Improvements

Security is mentioned but not discussed in detail in this thesis. There remain problems in how sandboxing and trusting should be integrated into WebMC to protect resources and user privacy.

In the current prototype, web-top applications access user’s PCE files via a file system interface. However, further exposure of CED through some other interfaces would provide applications with better flexibility and enable applications to explore the full power provided by CED.

In the current prototype, local resource registration is done through browser plug-in interface and an application description table. This is still not flexible enough and should be addressed in the new level of a WebMC middleware. Registration interfaces for applications could be discussed and proposed.

There are still places where the design of Additional Operations in CED could be improved. The current design is not flexible and somewhat protocol dependent. A better, more general approach is desirable.

Implementing the middleware in a stand-along process linked with Mozilla’s modules (as compared to building a Mozilla plug-in) will give us more flexibility and better control over display, caching and mimetypes.
6.2.2 New Directions

Personal computing environments of different users could be shared. Because WebMC uses CED as a central piece in PCE, so sharing of CED should be discussed in detailed. Issues like the unit of sharing, protection from unauthorized action could be addressed.

WebMC should better utilize emerging technologies for web-top applications. These technologies opened new possibilities for WebMC to provide more and better services to users.

A user’s CED could be adapted to be suitable on smaller devices with less displaying power. With the current separation of content and style, this could be considered natural and easy. Issues like choosing a display style according to the device’s display power should be addressed.

6.3 Conclusion

This thesis proposes a novel model for mobile computing called Web-based Mobile Computing (WebMC). The central idea behind this model is to use web languages to describe a user’s personal computing environment and extend the web browser to use this document to support user mobility. This model enables users to access user-defined computing environment from anywhere, at anytime and with any kind of Internet-connected computers.

WebMC is proposed to be a general solution to the current challenges in higher-layer mobile computing. As an important side-benefit, WebMC provides a generalized approach to customizable human computer interface design and generic application operating system interaction.

While WebMC described in this thesis along with the prototype demonstrated its feasibility and usability, it is only a starting point. Future research should be carried out to make this technology commercialized to meet user’s needs.
References:


*IEEE Personal Communications Magazine*, V 1, N 1, (First Quarter, 1994)

[Li1997] Yalun Li and et al, Protocol Architecture for Universal Personal 
Computing. *IEEE Journal on Selected Areas in Communications*. Vol 15, 
No 8, Oct 1997.

[Li1997-2] Yalun Li and et al. Supporting personal mobility for nomadic 
computing over Internet. *ACM Mobile computing and Communications 

[MicrosoftXML] Microsoft XML 2.5 SDK – XSL Developer’s Guide


Personal Communications*, Feb 2000.

Personal Mobility and Service Mobility in an Integrated Communication 
Network. *3rd IEEE Workshop on Mobile Computing Systems and 
Applications* (WMCSA2000), Monterey, CA, (December 2000).

management for high throughput computing. *Proceedings of the 7th IEEE 
International Symposium on High Performance Distributed Computing*, 


*Fifth ACM Symposium on Principles of Distributed Computing*, Man 
1996, Philadelphia, PA.


Appendix A: The Complete DTD of CED

<!-- ###########################################
The Document Type Definition of computing environment descriptor
Copyright by Naizhi Li 2000-2001
############################################# -->

<!ELEMENT CED (Userinfo, RootCollection)>
<!ATTLIST CED
  Version CDATA #REQUIRED
>
<!ELEMENT Userinfo (Name, Description, Supplementaryinfo)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Description (#PCDATA)>
<!ELEMENT Supplementaryinfo (#PCDATA)>
<!ELEMENT RootCollection (Collectioninfo, Self, (Documentpool | Servicepool | Preferencepool)*)>
<!ATTLIST RootCollection
  Name CDATA #REQUIRED
>
<!ELEMENT Collectioninfo (URLinfo?, AdditionalOperation?, Translation?)>
<!ELEMENT URLinfo (#PCDATA)>
<!ELEMENT AdditionalOperation (Createcollection?, CreateReference?, Update?, Deleteself?)>
<!ELEMENT Createcollection (Nameappendsimple | Nameappendinformserver)>
<!ELEMENT CreateReference (Nameappend | Nameappendsimple)>
<!ELEMENT Update (DefaultUpdate | SpecialURLUpdate)>
<!ELEMENT Deleteself (SpecialURLDelete)>
<!ELEMENT Nameappendsimple EMPTY>
<!ELEMENT Nameappendinformserver EMPTY>
<!ATTLIST Nameappendinformserver
  URL CDATA #REQUIRED
>
<!ELEMENT SpecialURLDelete EMPTY>
<!ATTLIST SpecialURLDelete
  URL CDATA #REQUIRED
>
<!ELEMENT Nameappend EMPTY>
<!ELEMENT DefaultUpdate EMPTY>
<!ELEMENT SpecialURLUpdate EMPTY>
<!ATTLIST SpecialURLUpdate
    URL CDATA #REQUIRED
>
<!ELEMENT Translation (#PCDATA)>
<!ELEMENT Self (ReferenceInfo)>
<!ATTLIST Self
    Name CDATA #REQUIRED
>
<!ELEMENT ReferenceInfo (MimeType, URL?, AdditionalOperation?, Translation?)>
<!ELEMENT MimeType (#PCDATA)>
<!ELEMENT URL (#PCDATA)>
<!ELEMENT DocumentPool (CollectionInfo, (DocumentPool | Document)*)>
<!ATTLIST DocumentPool
    Name CDATA #REQUIRED
>
<!ELEMENT Document (ReferenceInfo)>
<!ATTLIST Document
    Name CDATA #REQUIRED
>
<!ELEMENT ServicePool (CollectionInfo, (ServicePool | Service)*)>
<!ATTLIST ServicePool
    Name CDATA #REQUIRED
>
<!ELEMENT Service (ReferenceInfo)>
<!ATTLIST Service
    Name CDATA #REQUIRED
>
<!ELEMENT PreferencePool (CollectionInfo, (PreferencePool | Preference)*)>
<!ATTLIST PreferencePool
    Name CDATA #REQUIRED
>
<!ELEMENT Preference (ReferenceInfo)>
<!ATTLIST Preference
    Name CDATA #REQUIRED
>