A NEW ARCHITECTURE TO SUPPORT EFFICIENT WEB BROWSING IN A WIRELESS MOBILE COMPUTING ENVIRONMENT

BY

SCOTT D. WALKTY

A Thesis
Submitted to the Faculty of Graduate Studies
in Partial Fulfillment of the Requirements
for the Degree of

MASTER OF SCIENCE

Department of Computer Science
University of Manitoba
Winnipeg, Manitoba

0-612-53238-0

A New Architecture to Support Efficient Web Browsing in a Wireless Mobile Computing Environment

BY

Scott D. Walkty

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University

of Manitoba in partial fulfillment of the requirements of the degree

of

Master of Science

Abstract

Mobile computing environments present several challenges arising from the limitations inherent in wireless networks and the inability of current network protocols to cope with these limitations. As the popularity of untethered access to the Internet increases, conventional network applications such as web browsing must overcome these challenges to support the needs of mobile users. Recent work in the mobile computing research community has attempted to increase the efficiency of web browsing in a mobile computing environment by employing a client/intercept/server architecture which enables the optimization of application layer data transmitted across the wireless portion of the network. Although effective, this strategy relies on the existence of a wired side agent, which introduces a new problem with respect to mobility across different heterogeneous networks (where the agent may or may not be available). This thesis presents a new architecture to support the optimization of web browsing in a wireless mobile computing environment. The architecture uses mobile agents to dynamically deploy a client/intercept/server architecture on foreign networks that provide a certain mobility framework.

The new architecture offers the following specific advantages: (1) A framework is identified that enables mobile units to discover and use mobile agent systems on foreign networks; (2) The benefits of a client/intercept/server architecture are translated to any network that supports this framework; (3) The architecture works with existing Internet protocols; (4) Mobile agent systems on foreign networks are modeled as a service that the foreign network provides, similar to other services such as printers.

Finally, a prototype system is described that is implemented using the architecture to demonstrate the feasibility of the approach.

Acknowledgements

This thesis would not have been possible without the assistance and inspiration of several individuals. I would like to offer my sincere thanks to both my advisors, Dr. Randal Peters and Dr. Ken Barker, for their guidance and support. This work grew out of a course offered by Dr. Peters and both he and Dr. Barker have provided continuous assistance in the face of difficult working conditions. They allowed me the freedom to follow my own path while offering the right amount of guidance to keep me on track. Thanks also go out to the National Science and Engineering Research Council and the University of Manitoba for the financial assistance they provided.

I would like to thank my friends for their support and for helping to keep things in the proper perspective when the workload became difficult.

Finally, I would like to express my deepest thanks to my family; my parents Ian and Kathy Walkty and my brother Andrew. Their constant encouragement, support and inspiration have made this work possible.

Contents

List of Tables

List of Figures

Chapter 1

INTRODUCTION

1.1 Overview

The integration of fixed and mobile computers via wireless networks has witnessed the emergence of mobile computing environments promising to deliver network capability from anywhere, anytime. It has been predicted that mobile access will be the third killer app for the Internet after email and web browsing [Nei98]. However, for such a prediction to become a reality the limitations imposed by wireless networks and mobile computing environments must be overcome to support popular network applications such as web browsing.

Conventional network applications encounter several problems when applied in a mobile computing environment. These applications were designed to operate on reliable, high bandwidth, wired networks and are unable to cope with the challenges presented by wireless environments. This thesis is specifically concerned with the application domain of web browsing. Difficulties facing web browsing in a mobile computing environment stem from two sources: the limitations inherent in wireless networks and the inadequacies in the HTTP protocol when applied in such networks. In general wireless networks can be characterized as having high cost, high latency, low bandwidth, and unreliable connections when compared with their wired counterparts [Hou96]. Moreover the HTTP protocol exacerbates these problems as it is a verbose protocol with high connection overhead and internal fragmentation [Che99, Hou96]. HTTP was never intended to work within the low bandwidth environments presented by wireless networks. These issues are well known in the mobile computing research community and a good deal of work has been performed on constructing systems that are able to support efficient web browsing given the limitations of wireless networks and HTTP by performing optimizing operations at the application layer. Typically these systems employ a form of proxy architecture, where the proxy attempts to alter the application layer data transmitted across the wireless network to achieve better efficiency.

A popular approach used to support web browsing in mobile computing environments is the deployment of a client/intercept/server architecture [Baq95, Lil95, Hou96]. This is essentially a distributed 2-proxy architecture where one proxy resides on the mobile unit and the other proxy resides on a fixed unit within the wired network. The proxies encapsulate transmissions that take place over the wireless link. Communications between the mobile unit and the fixed network go through the proxies. This architecture offers the advantage that the two proxies have complete control over the type and format of the application layer information flowing across the wireless link and can perform optimizations at this layer to increase the efficiency of the transmissions. Furthermore, optimizations can be performed in either direction. However the client/intercept/server architecture suffers from a major limitation: the reliance on a wired side component. This reliance hinders mobility since optimizations can only be realized on networks that contain the wired side proxy or permit remote access to the proxy. The assumption that a fixed wired side proxy will be accessible on an arbitrary network is invalid given the heterogeneous nature of the networks encountered by a mobile user.

1.2 Scope and Contributions

This thesis describes a new architecture designed to support efficient web browsing in a mobile computing environment. The architecture presented here employs mobile agent technology to dynamically deploy a client/intercept/server system onto a foreign network. Once in place, the system performs optimizations at the application layer in an attempt to increase the efficiency of the communications. The architecture was designed to work within the existing Internet infrastructure and to enable the effective use of client/intercept/server systems while placing few constraints on the foreign networks (refer to Sections 1.2.1 and 3.2 for details on the constraints imposed) with which a mobile unit communicates. This thesis also describes the design and implementation of a prototype system that has been developed to demonstrate the feasibility of the architecture. The prototype system was rigorously tested to assess the correctness of the implementation with respect to operation and behavior of the different architectural components. Performance issues regarding the optimizations employed by the prototype were not addressed in this thesis. Both the architecture and implementation are compared with existing architectures and systems to highlight the novel features of the new architecture.

The architecture presented in this thesis is composed of two major pieces: the mobility framework and the mobile web browser support system. This section describes the contributions of these architectural components and explains the relevance of the architecture as a whole when applied to domains other than web browsing. Note that the architecture described in this work does not in itself overcome the problems of web browsing in a wireless environment. Rather the architecture is designed to allow the deployment of a system that can perform application layer optimizations in an attempt to make web browsing more efficient. Hence the architecture *supports* efficient web browsing, but does not directly yield a more efficient solution. Increased efficiency (or lack thereof) is a result of specific optimizations that are employed in an implementation of the architecture.

1.2.1 Mobility Framework

The mobility framework identifies a set of services that could be provided by foreign networks capable of hosting mobile agents for visiting mobile units. Many researchers have noted the benefits of mobile agents in a mobile computing environment [Jos99b, Jos97, Sah98, Gra96], but overlook the issue of how to employ agents in a network where the location of network services is not known in advance (i.e. the locations of the execution environments for the mobile agents are not known *a priori*). This point is important as uncertainty about the environment arises as a natural consequence of the heterogeneous networks with which a mobile user communicates. The mobility framework solves this problem and offers the following specific contributions:

- The set of services that a mobile unit requires from a foreign network to support the deployment of mobile agents onto the foreign network are explicitly identified.

- The services employed by the mobility framework require no changes to the existing Internet protocols.

1.2.2 Mobile Web Browser Support System

The mobile web browser support system component of the architecture uses the mobility framework to dynamically deploy a client/intercept/server architecture onto a foreign network. Such an architecture is capable of performing a wide range of operations to try to optimize the application layer data flowing across the wireless communications link. The prototype implemented demonstrates a number of these operations including compressing data before it is transmitted, adjusting the fidelity of images in response to changing communications link quality and the use of a user profile to select the preferred type of data. In addition to the benefits a client/intercept/server architecture already offers mobile computing environments, the mobile web browser support system makes the following contribution:

- The benefits of the client/intercept/server architecture are translated to any network that supports the mobility framework (refer to Sections 1.2.1 and 3.2 for additional details).

The dynamic deployment of the client/intercept/server architecture has a distinct advantage over existing architectures to support mobile web browsing since it does not require the existence of a proprietary wired side component on networks with which the mobile unit interacts. The proprietary pieces of the architecture are initially resident on the mobile unit. This reliance is not accounted for in the other systems using a client/intercept/server architecture, but is important when the unit is mobile and communicates with several heterogeneous networks. These networks almost certainly will not all contain the proprietary wired side component and in some cases will prevent the user from remotely accessing the wired side component (e.g. in the case that the foreign network is protected by a firewall). By dynamically deploying the wired side part of the client/intercept/server architecture from the mobile unit rather than using a separate program installed on the fixed network the architecture becomes more flexible. The reliance on a wired side proxy is replaced by a reliance on the mobility framework. As the mobile unit changes location, it takes with it the components required to set up a client/intercept/server architecture. Thus the benefits of the architecture are accessible on any network with which the mobile unit communicates, subject to the assumption that the mobility framework is supported.

1.2.3 Relevance

Although the work described in this thesis is limited to the context of web browsing the ideas presented in the architecture could be extended to other network applications executing in a mobile computing environment.

The mobility framework identifies the services that are necessary to support the deployment of mobile agents onto a foreign network where the location of the mobile agent execution environment is unknown. This framework enables the insertion of a wide variety of application layer proxies into a foreign network. Moreover the framework allows other types of mobile agents to be dispatched onto the foreign network. Agents that perform diverse tasks such as executing queries on remote databases located on the foreign network could be transmitted as easily as an agent acting as a proxy for the mobile unit. The framework supports the use of these agents by providing a means to connect to the network, and to discover and use the mobile agent service.

The mobile web browser support system describes the dynamic deployment of a client/intercept/server architecture onto a foreign network in a mobile computing environment. At a high level the goal of the support system is to encapsulate a proprietary protocol that optimizes application layer data across a wireless communications link. In the case of this thesis, the HTTP protocol (used for web browsing) is targeted for optimization. However the architecture could potentially be applied to other application layer protocols as well.

1.3 Organization

The remainder of this thesis is divided into five chapters that provide further details on the problem area and related work. They describe the components of the architecture and the prototype implementation in depth.

Background information and related work are covered in Chapter 2. The chapter begins with a discussion of the problem area identifying the key concepts and terminology in the field of mobile computing necessary to understand the remainder of the thesis. The limitations of wireless networks and the HTTP protocol applied in a mobile computing environment are described in detail as a motivation for the work presented here. Furthermore, background information is provided on the operation of proxy architectures in general and the client/intercept/server architecture in particular to provide a better understanding of the related systems and the new architecture described in the thesis. A number of enabling technologies for mobile computing are also described to give a solid foundation in the protocols that are used in the new architecture and the prototype implementation. Included in this discussion are protocols for mobile network connectivity (DHCP, Mobile IP and IPv6 mobility support), protocols for dynamic service discovery (resource discovery protocol and SLP) and an introduction to the key concepts of mobile agents and agent systems. The chapter concludes with a description of current research in mobile computing that relates to the architecture and prototype presented in this thesis. Comparisons between the systems described in Chapter 2 and the architecture presented here are deferred until Chapters 3 and 4.

In Chapter 3, the new architecture developed in this thesis is introduced and described in detail. Definitions are covered that pertain to the architecture and the design goals of the architecture are outlined. Both the mobile web browser support system and the mobility framework are examined individually and with respect to their interaction. The chapter then provides a comparison between the new architecture and the architectures employed in the existing systems identified in Chapter 2. Chapter 3 ends with a summary of the advantages the new architecture offers over other existing architectures.

The implementation of a prototype mobile web browser support system based on the architecture described in Chapter 3 is presented in Chapter 4. The hardware on which the system runs is identified and the implementation of the prototype is explained with respect to the different components of the architecture. Specifically, information is provided on the configuration of the foreign network to support the mobility framework and the operation of the mobile web browser support system. The operations employed by the support system to optimize the application layer transmissions are also examined along with the communication protocol used between the proxies in the client/intercept/server architecture. This chapter provides a brief description of some implementation issues arising with respect to incorporating the Mobile IP protocol into the mobility framework. Finally, a comparison is presented between the prototype and the implementations of related systems. The comparison is performed mainly with respect to the optimizing operations employed since the different architectures are compared in Chapter 3.

Chapter 5 describes the process used to test the prototype implementation for correctness and the tuning required based on the initial results of system testing. The test plan used to evaluate the system is outlined and the results of the tests are discussed. The chapter concludes by describing the problems that were uncovered and the steps that were taken to correct them.

Conclusions and contributions of the thesis are presented in Chapter 6. The chapter summarizes the work done, lists the contributions and provides some suggestions of future research directions.

There are three appendices included at the end of the thesis. Appendix A provides the service type templates used by the Service Location Protocol (SLP) in the mobility framework to identify the mobile agent system service on foreign networks. Appendix B describes in detail the type and format of messages in the proprietary protocol used between the proxies in the mobile agent support system. Finally, Appendix C contains a copy of the test plan that was followed when evaluating the operation of the prototype system.

Chapter 2

BACKGROUND

This chapter presents a broad overview of the background material on which the thesis builds. The general concepts and enabling technologies of mobile computing are explored in detail as an introduction to the problem area. An overview of the state of the art in mobile agent technologies and agent systems is also presented, including definitions and key concepts. Following this, a selection of related work on the subject of mobile web browsing is highlighted to provide a frame of reference against which to compare the work presented here.

2.1 Problem Domain

The first section gives background on the problem area considered in this thesis. Mobile computing is defined and the challenges arising from mobile computing (in particular those pertaining to web browsing) are discussed. Limitations of the HTTP protocol are presented as a further motivation for this work, and general proxy architectures, which provide a foundation for much of the related work, are described.

2.1.1 Mobile Computing

Mobile computing is the ability to provide computing and networking capability from anywhere, at any time [Pet99]. Central to this paradigm is the existence of mobile computing environments. A mobile computing environment consists of a fixed (i.e. wired) and/or wireless network. So we have two types of computers: fixed units and mobile units. Fixed units (FU) are computers whose location remains stationary (e.g. a typical desktop computer) while mobile units (MU) are computers whose geographic and network location can easily change (e.g. a laptop computer or a personal digital assistant)

[Pet99]. Typically the mobile units communicate with the fixed units and each other using wireless LAN/WAN technologies, which facilitate mobility. Mobile computing enables a user to access network services (such as web browsing) irrespective of physical and/or network location. It is important to make the distinction between portability and mobility [Per98]. Mobility refers to the seamless roaming of a mobile unit between networks. When mobility is supported, the mobile unit can move from one network to another without losing connectivity. Moreover, it is able to use a single network address regardless of the current point of attachment. On the other hand, portability generally implies that the mobile unit receives a new address each time it connects to a different network. When a portable node moves from one network to another, all network subsystems must be restarted and a new address must be obtained. Portability is most suitable when the change in network attachment overlaps points in time where the mobile unit is disconnected or turned off [Per98]. Although this distinction exists, the techniques discussed in this thesis will be relevant to mobile computing environments that support either portability or mobility.

It should also be noted that wireless network access is not the same as mobility [Per98b]. Wireless networks are a technology that enables mobility. Although mobile computing environments typically rely on wireless networks, wireless network access is not a necessary condition for mobility. It is possible for computers that are stationary (i.e. not mobile) to communicate with each other over wireless networks. Likewise, it is possible to have mobile laptops communicating over a wired interface (e.g. unplug a laptop from a network at a university, transport it to an office building, and plug it into the office network), hence mobility without wireless access. This thesis will only be concerned with mobile computing environments that rely on wireless networks, since the problems related to web browsing in mobile computing environments arise mainly from the limitations imposed by wireless communication links. In mobile environments that rely on fixed (wired) connections, web browsing presents no special difficulties with respect to bandwidth availability.

There are two possible configurations of wireless mobile computing environments. The first configuration (Figure 2.1) is the most common, where wireless mobile units communicate with a fixed (wired) network via wireless access points (basestations) attached to the network. In this configuration, the wired network provides the basic services that the wireless user accesses. Generally speaking, the mobile unit is a client, connecting via a wireless link to the servers on the network. This configuration is fairly well understood in terms of providing services to mobile clients and partitioning of client/server responsibilities. The second configuration (Figure 2.2) is referred to as *ad-hoc* wireless networking. *Ad-hoc* wireless networks consist of a number of wireless mobile units that dynamically form a network for communications purposes [Per98b]. This configuration raises a number of issues such as how routing and addressing are performed. *Ad-hoc* wireless networks are beyond the scope of this thesis. The focus here is entirely upon wired networks that support mobility by providing wireless access to mobile clients.

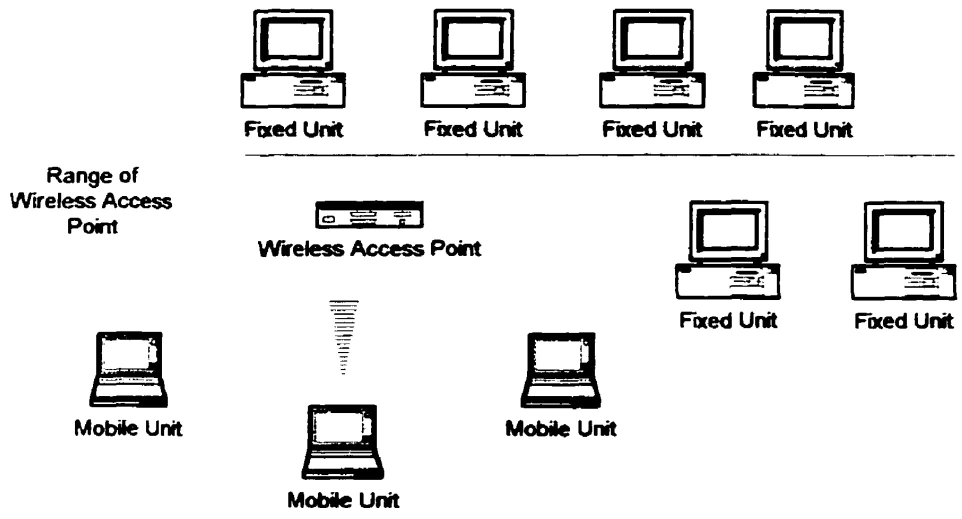


Figure 2.1 – Fixed Network with Wireless Access Points

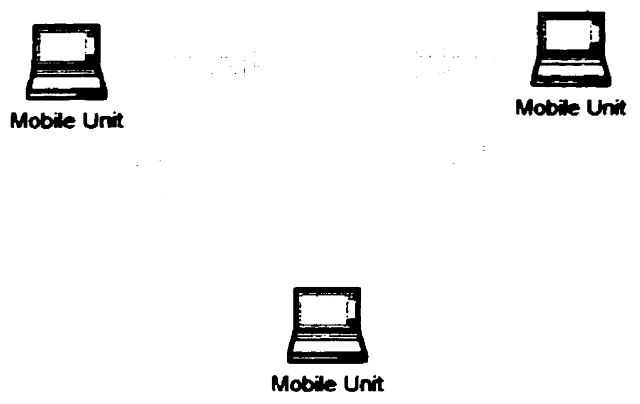


Figure 2.2 - Ad-hoc Wireless Network

Mobile computing environments give rise to a number of challenges that are not encountered in traditional networked environments [For94][Sat96]. Mobile clients are traditionally resource poor when compared to their fixed counterparts since they are intended to be portable (hence smaller and lightweight). They are characterized by limited display area, processor speed, memory and disk capacity. Furthermore, battery power is a finite resource and must be managed carefully. Issues pertaining to the location of the mobile user also arise, with respect to determining the location of the mobile unit, and forwarding information destined for the unit to that location [For94]. Finally, there are a host of challenges that crop up from the usage of wireless networks as an enabling technology for mobility. It is these challenges that are particularly relevant to the application domain of web browsing and special attention should be paid to them. Wireless networks may generally be characterized by the following inadequacies [Hou96]:

- High Cost – Users are often charged for wireless network access by the amount of time connected or the number of bytes transmitted. Furthermore, the cost per byte transmitted is much higher than for wired networks.
- Low Bandwidth - The capacity of wireless networks is very limited when compared to wired networks. Typical bandwidths supported by wireless connections are in the neighborhood of 1-2 Mbps (e.g. WaveLAN) and can go as low as around 4800 bps (e.g. Ardis) [Hou96]. Recent advances in wireless networking have yielded interfaces capable of supporting up to around 11 Mbps (e.g. Orinoco PC card, formerly

WaveLAN Turbo 11 Mbps and Apple Airport) [Air99][Wav99]. However, wired networks commonly support speeds of around 10 Mbps to 100 Mbps. As wireless network speeds are increasing, so too are wired network speeds with fiber optic networks achieving speeds in the gigabit range [Tan96]. Although wireless network speeds continue to improve, there will likely always be a large gap in their speed as compared to fixed networks [Jos99].

- Unreliable Connections – Mobile devices are subject to frequent, unforeseeable disconnections as they move out of range of the wireless network or behind barriers that block the network signal.
- High Latency – Due to the low bandwidth, retransmissions (necessary because of the unreliable connections), error processing control (necessary because of higher error rates caused by the less reliable wireless connections) and other factors, the response time for wireless links is necessarily very high (much more so than for their wired counterparts). [For94]
- Security Risks – Due to the ease of connecting to a wireless network. Any unit with a receiver tuned to the proper radio frequency used by a transmitting node can intercept the information that is being sent (particularly if the transmission range covers a large area) [For94].

The security risks are not quite as important in the web browsing domain, since most web pages are accessible to the general public (except in the case where secure information, such as credit card numbers, is being transmitted). However, the other four limitations must be overcome to perform web browsing effectively within a mobile computing environment.

One other important challenge that must be overcome when working in a mobile computing environment is the high variability that these environments represent. The source of this variability is due to the nature of wireless connections (as discussed above, disconnections occur frequently) and the interaction of the mobile unit with heterogeneous networks [For94]. Most stationary computers (fixed units) will remain connected to a single network, providing a reliable connection and a relatively fixed bandwidth (subject to other traffic on the network). On the other hand, mobile computers encounter a variety of network configurations (e.g. variability in the available bandwidth, number of users, etc.) as they move out of the range of one network and into the range of another. These networks may differ in a number of ways, and variability arises with respect to bandwidth, latency and cost [Nob95] [Wel97].

This variability gives rise to one of the key concepts in designing applications that run in mobile computing environments: adaptability [Sat96] [Nob95] [Wel97]. Applications should be able to adapt to changing network conditions and environments, to make the most effective use of the available resources (e.g. bandwidth). The mobile unit must be able to dynamically conform to whatever limitations are presented by the current environment [Nob95]. Network protocols that enable a mobile unit to transparently maintain connectivity across networks (e.g. Mobile IP, see Section 2.2.2) largely ignore the changing link characteristics. Hence it is important that the mobile applications be able to recognize current conditions and adapt accordingly as available resources deteriorate [Wel97]. Strategies for adaptation may be classified as either *laissez-faire* (in which the application performs all the adaptation without any system support), *application-transparent* (in which the system performs the adaptation) and *application-aware* (where the application determines how to adapt, and the system monitors resources and exposes properties to the application that allow it to make that determination) [Sat96]. It has been noted that in many cases application-transparent adaptation is desirable because it is not feasible to force users to purchase new mobile aware applications [Per98]. Web browsing applications fall into this category, since web browsers have traditionally not been developed with mobility in mind.

2.1.2 HTTP

HTTP1.0 [Ber96] is the standardized application layer protocol used by commercial web browsers to retrieve HTML entities (such as web pages or images) from a web server. It is a simple request/response protocol [Ber96] where the client establishes a separate connection to the web server for each entity that is requested. Once a connection is set up, the client uses an HTTP request message to request a specific entity from the server (specified in the message by the URL, Uniform Resource Locator, of the entity). The server replies with a response message containing a status code and the requested entity.

Following the response message, the connection is terminated. HTTP 1.0 is stateless [Lil95] in the sense that it does not maintain a connection beyond the receipt of a response from the server. No information is stored regarding the connection. Each HTTP connection runs on top of a separate TCP connection from the client to the web server.

HTTP 1.0 supports a number of different request operations, called methods [Ber96]. The most common of these are GET, HEAD, and POST. A GET request (the most common type of request in web browsing) is used to retrieve a specific entity from a web server. GET requests are used by web browsers to retrieve most web pages presented to the user. A HEAD request (not very common) is similar to a GET request, but is used to retrieve just the header part of the response message. It can be used to find information about the requested entity (such as the entity size or expiration time) without actually downloading the entity to the client. The third type of request, POST, is used to transmit information to the web server. POST requests are sometimes employed on web pages that contain HTML forms. Once the form is filled out, the data entered by the user is submitted to the web server via a POST request. The data itself is contained in the body part of the request message. The GET and HEAD request messages typically consist of a header only (indicating the URL of the requested entity) with an empty body. Response messages always follow the same format, with a status line (indicating the protocol version and a success or error code), a set of optional header fields and the entity body (containing the requested entity, if it could be retrieved) [Ber96].

HTTP 1.0 works fairly well when deployed in the high bandwidth, low latency fixed networks for which it was designed. However, there are a number of problems that have been noted. Principal among these is the requirement that HTTP 1.0 open a separate connection for each object that it requests (in particular, each object on a web page) [Che99]. This requirement introduces a large amount of connection overhead (with respect to setting up and tearing down the connections), which has a detrimental effect on network throughput and load [Che99]. Also, because most HTTP 1.0 connections are short lived (due to the relatively small size of the requested entities) and TCP uses a slow start protocol to gradually increase the rate at which it transmits data (thereby avoiding congestion) the connection will often never reach its full throughput. Furthermore, HTTP 1.0 causes high internal fragmentation. If an HTTP entity (contained in an HTTP response message) is slightly larger than can fit into a single packet, then two packets must be transmitted. The first packet will be full, but the second packet may contain only a few bits. Since both packets carry a 40 byte header (20 bytes for the IP header and 20 bytes for the TCP header), the second packet wastes additional bandwidth [Che99]. Also both router and CPU resources will be wasted trying to process this packet (which contains only a few bits). However, due to the use of separate connections, it is not possible to include the remaining few bits from one entity in a packet containing another entity. Hence the small packets are necessary to transfer the complete entity. These problems are further exacerbated when the HTTP 1.0 protocol is applied in low bandwidth, high latency wireless mobile computing environments.

When considering HTTP 1.0 in a wireless mobile computing environment, there are four major problems that arise [Che99][Hou96]:

- Connection Overhead – Due to the opening of a new TCP connection for each object requested by HTTP 1.0 (as discussed above)
- High Internal Fragmentation – Also discussed above. A direct result of opening a separate connection for each requested entity [Che99].
- Redundant Header Information – HTTP 1.0 request messages transmitted by the web browser generally include redundant header information (around 200 to 400 bytes) specifying the capabilities of the browser (e.g. the different content types that it supports). Since the web browser application is relatively stable during a browsing session, retransmission of this information with each request is redundant and unnecessary in a bandwidth-limited environment [Hou96].
- Verbose Protocol – HTTP messages are encoded in standard ASCII so as to be human readable. As with the redundant header information, this wastes bandwidth when the information could be encoded in a more compact, yet less readable format [Hou96].

To some degree, these problems are mitigated with the standardization of HTTP 1.1 [Fie97]. HTTP 1.1 was designed to address the problems arising from HTTP 1.0. HTTP 1.1 allows for the use of persistent network connections and pipelining requests over the connections [Fie97][Che99]. This reduces the amount of connection overhead experienced by HTTP 1.0 (fewer connections have to be set up, since one connection can be used to retrieve multiple objects) and alleviates the internal fragmentation problem to a degree (since multiple responses may be transmitted along the same connection, due to the pipelined requests, and hence may be packed together). However, HTTP 1.1 has not yet gained widespread adoption in the Internet community. Many commercial web browsers (e.g. Netscape) still employ HTTP 1.0 to retrieve information from web servers.

2.1.3 Proxy Architectures

Recently a good deal of research has focused on the use of wired side proxies to overcome the limitations imposed by mobile computing environments at the application layer (See Section 2.4). A proxy works as an intermediary between a client and a server. The proxy itself implements both the client and server functionality for the particular protocol that it supports. Classical proxies operate in the following way [Cha96]. Clients initiate a connection to a server, which is accepted by the proxy. The proxy appears as the server to the client. The proxy proceeds to open a separate connection to the server, and appears as the client to the server. After establishing the necessary connections, the proxy is used to relay requests and responses between the client and the server. The functional goal of the proxy architecture is to relay information between clients and servers that do not have direct IP connectivity.

Proxies are of interest in mobile computing environments because of the limited resources available to the mobile units and the rich resources available to the fixed units with which they are communicating. Placing a proxy on a fixed unit inside the wired portion of the mobile computing environment allows the mobile units to offload some of their processing to the fixed network. The proxies employed in mobile computing environments are more specialized than the classical proxies described above. They generally provide functionality beyond that of classical proxies (which are often used for caching or filtering information). Specific advantages that proxies bring to mobile computing environments include, but are not limited to [Per98b]:

- Power Savings – By offloading processing to the fixed network, the CPU on the mobile unit performs less processing which lessens the burden on the power supply.
- Performance Improvements – This includes better use of available bandwidth (since a proxy may perform such operations as compressing data before it is transmitted) and

improvements in speed. Speed increases because the proxy can perform some negotiations with the server on behalf of the mobile client directly on the fixed network.

- Disconnected Operation – The proxy can continue to perform operations on behalf of the mobile client while the client is disconnected.

Due to their ability to modify the data that passes through them without requiring any changes to either of the end systems (client or server), proxies are often used as a mechanism to support application-transparent adaptation. Consider the application domain of web browsing. *The responsibility for the adaptation used to overcome the limitations of wireless networks may rest solely on the proxy with which the web browser communicates.* This enables conventional web browsers to work effectively in a mobile computing environment and allows the web browser to be independent of the adaptations employed. Wired side proxies have been successfully applied to support mobile web browsing by incorporating functionality such as removing HTML tags that a mobile unit does not support [Jos99], reducing the size of images [Jos99], removing active content (e.g. Java applets) from web pages [Jos99], etc.

2.1.4 Client/Intercept/Server Architecture

In mobile computing environments, one useful extension to the basic proxy architecture described above has been the incorporation of a second proxy (Figure 2.3), residing on the mobile unit [Baq95, Lil95, Hou96]. Such a two proxy architecture will be referred to here as a client/intercept/server architecture, so named for its behavior. Web browsing using such an architecture occurs in the following manner. The proxy operating on the mobile unit intercepts HTTP requests from the web browser. This proxy forwards the requests to the proxy on the fixed unit, which communicates with the web server to retrieve the requested entity. The proxy on the fixed unit then forwards the response back to the proxy on the mobile unit, which in turn transmits the response to the web browser. The major advantage of such an architecture is that the two proxies have complete control over the type and format of the application layer information passing over the wireless link. Moreover, this control is bi-directional (whereas in a single proxy scheme, application layer optimizations may only take place from the fixed unit to the mobile unit). Placing a proxy on either end of the wireless link allows certain optimizations that require an inverse operation to be used, while maintaining application transparency (note that although there is transparency with respect to the operation performed, there is not complete transparency for the user since the browser must still be configured to use the proxy on the mobile unit). For example, if a page is compressed on the fixed unit, it must be possible to decompress the page on the mobile unit. Not all browsers may support the compression technique used by the wired side proxy, so the mobile side proxy must be used to decompress the data.

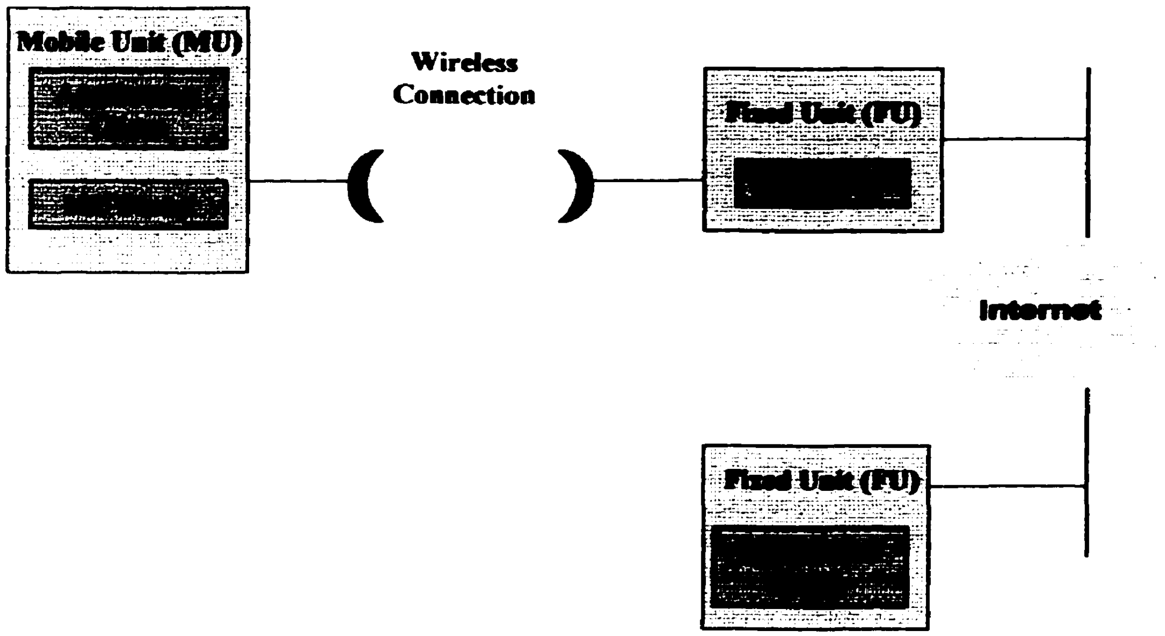


Figure 2.3 - Client/Intercept/Server (2 proxy) Architecture

2.2 Enabling Technologies

A basic knowledge of the enabling technologies of mobile computing is key to understanding the architecture presented in this thesis. This section gives an overview of these technologies, focusing on network protocols for providing connectivity to mobile units, and issues of service discovery.

2.2.1 DHCP

To route packets to a host connected to the Internet, the host must be configured with a valid IP address for its current point of attachment (subnet). Once configured with this address, packets may be transmitted to the host using the regular Internet routing infrastructure. The Dynamic Host Configuration Protocol (DHCP) [Dor93] [Wim93] provides a mechanism for dynamically assigning an IP address to a host, along with a number of additional configuration parameters including subnet mask, default router, DNS server and domain name. The IP address provided via DHCP is valid for a limited period of time only, known as a lease. Hosts may extend the lease through subsequent DHCP requests, but are not allowed to use the IP address after the lease has expired.

DHCP [Dro93] [Wim93] is a straightforward extension to the BOOTP protocol [Cro85] (used to dynamically provide an operating system image to diskless workstations over a network). DHCP messages are carried within BOOTP; BOOTREQUEST and BOOTREPLY messages. As with BOOTP, DHCP uses UDP as a transport layer protocol. The operation of the DHCP protocol is as follows. A DHCP client (i.e. a host that wishes to obtain an IP address on the subnet to which it is currently connected) broadcasts a DHCPDISCOVER message on the local subnet to the well know DHCP server UDP port (67). BOOTP relay agents may forward these requests if there is no DHCP server on the local subnet (DHCP servers can be configured to provide addresses for subnets other than the one to which they are directly connected). Any DHCP server that receives the DHCPDISCOVER request may respond with a DHCPOFFER message to the client, which includes an available IP address. The client selects one address from all the offers it receives and broadcasts a DHCPREQUEST message, containing the IP address of the DHCP server whose offer it has accepted (along with optional values to specify any additional configuration parameters that are required). The chosen server recognizes its IP address in the request message and responds with a DHCPACK message, containing the offered IP address and the requested configuration parameters. All other DHCP servers view the request as a notification that the client has declined their offers.

DHCP seems to be an attractive means of providing connectivity to mobile units through its dynamic assignment of IP addresses. Both [Per95] and [Per95b] explored the possibility of incorporating DHCP within mobile computing environments. However, when used as described above, DHCP only provides support for portability (as defined in Section 2.1.1) rather than true mobility. Each time the mobile unit moves to a new network, it must use DHCP to obtain another IP address. This cannot be accomplished transparently to the user since DHCP has no facilities to detect a change in the point of attachment. Hence the user must manually force DHCP to obtain a new IP address via a network configuration utility (for example the winipcfg program on Windows) whenever the mobile unit encounters a new network. Furthermore there is no way to maintain network connections between different points of attachment because the IP address of the mobile unit changes with each network. Therefore DHCP does not support seamless roaming (mobility) between networks.

With respect to mobile computing, DHCP offers the advantages that it is inexpensive, simple and can be implemented entirely on the network with which the mobile unit is communicating (it requires no additional support from the mobile unit's home network). Nevertheless, there are two major disadvantages to this approach. As mentioned previously, only portability is supported. Moreover, DHCP does not provide a mechanism that allows other computers to actively locate the host; it only includes the host in the routing infrastructure. This is suitable for applications such as web browsing, which operate under a request/response protocol since the request transmitted to the web server will contain the return IP address (allowing the server to locate the client). However, for mobile units acting as servers (e.g. a web server) this method is not feasible since there is no way for other hosts in the Internet to actively determine the current IP address used by the mobile unit. Furthermore, DNS entries (which map the host name to its IP address) for the mobile unit will be invalidated each time the mobile unit receives a new IP address. The second problem associated with using DHCP in mobile computing is security. The DHCP protocol [Dro93] itself makes no provision for security. This problem is exacerbated in wireless networks, which must already contend with challenging security issues. DHCP is vulnerable to a number of attacks including denial of service attacks (in which a malicious host could potentially request all available addresses) and server impersonation (where a host could masquerade as a DHCP server and offer bad Internet addresses) [Per95].

In [Per95] and [Per95b], the authors also investigated the possibility of using DHCP in conjunction with the Mobile IP protocol. The results of this application of DHCP in a mobile computing environment were more promising.

2.2.2 Mobile IP

Mobile IP [Per96] [Sol98] is a network protocol, which was designed to support the transparent routing of IP datagrams to mobile nodes. It typically (but not always) operates over the IP protocol. Similar to DHCP, it provides a mechanism by which a mobile node may connect to a foreign network. However, unlike DHCP, which supports only portability, Mobile IP offers true mobility support with seamless roaming between various networks. Moreover, it allows mobile nodes to continue to use their home IP address in communications, irrespective of their current point of attachment. This feature enables the deployment of Mobile IP without requiring any changes to correspondent nodes with which the mobile node is communicating. A detailed description of the mobile IP protocol is available in [Per96], [Per98], and [Sol98]. The following paragraphs will present a brief overview, explaining the major components and operation of the protocol.

RFC 2002 [Per96] defines a number of terms that are useful when discussing the operation of Mobile IP:

- Home Network – The home network for a mobile node is the network that has a network prefix (in the network IP address) matching the network prefix of the mobile node's IP address.
- Foreign Network – Any other network that the mobile node encounters.
- Home Agent – A router on the mobile node's home network. The home agent is responsible for tunneling packets destined for the mobile node to the mobile node's current "care of address" (when the mobile node is not connected to the home network).
- Foreign Agent – A router on the foreign network. It is the responsibility of the foreign agent to provide routing services to visiting mobile nodes that are registered with the foreign agent. It is also the job of the foreign agent to detunnel and deliver IP

datagrams to the mobile node that have been tunneled to the foreign agent by the mobile node's home agent.

- Care of Address – Used to identify the end point of a tunnel to the mobile node. There are two types of care of address: (a) a foreign agent care of address (i.e. the IP address of a foreign agent) is used when the mobile node is registered with a foreign agent. (b) a co-located care of address is used when the care of address is obtained by some external mechanism (e.g. DHCP) and is assigned to one of the mobile node's network interfaces.

Mobile IP functionality is organized into three components: agent discovery, registration and routing mechanisms. Agent discovery is the process by which mobile units may determine their current location, and obtain a care of address. Registration is the process by which a mobile unit informs its home agent of its current care of address and requests routing services from a foreign agent. Routing mechanisms specify how datagrams are delivered to the mobile unit via Mobile IP. The basic operation of Mobile IP consists of the following steps [Per96] [Sol98]:

- Foreign and Home agents periodically advertise their presence on a network via agent advertisement messages. Mobile units use these messages to determine whether they are connected to their home network or a foreign network. A mobile unit may solicit these advertisements if they are not forthcoming (e.g. if a mobile unit is trying to detect if it has moved to another network).
- As long as the mobile unit is attached to its home network, it operates in the normal fashion. One exception to this is when it first returns home after visiting a foreign network. In this case, it informs the home agent of its return (deregisters with the home agent) before resuming normal operation.
- When a mobile unit detects that it has moved to a new network, it obtains a care of address, either from a foreign agent on the network or via some external mechanism (such as DHCP). The mobile node proceeds to register the care of address with its home agent via registration request and reply messages. The purpose of this registration is to inform the home agent where to forward datagrams for the mobile node. If the mobile node is using a foreign agent care of address, the registration takes

place through the foreign agent. This serves the secondary purpose of registering the mobile node with the foreign agent, so that the foreign agent is able to deliver datagrams to the mobile node that it receives from the mobile node's home agent.

- IP datagrams transmitted from the mobile node are routed via the usual IP routing mechanisms. They contain the mobile unit's home IP address as the source address (as mentioned previously, the mobile node is able to continue to use its home address, regardless of point of attachment).
- IP datagrams transmitted by correspondent nodes will naturally be routed to the mobile node's home network (in an attempt to deliver them to the mobile node's home address). On the home network, these datagrams are intercepted by the home agent and tunneled to the mobile unit's current care of address (tunneling is performed by encapsulating the IP datagram inside another IP datagram addressed to the mobile node's care of address). The receiving foreign agent (or mobile node, if a co-located care of address is used) detunnels the datagram (i.e. removes the inner IP packet) and delivers it to the mobile node.

The main benefit of this approach is that it supports seamless mobility without requiring any changes to correspondent nodes. However, several drawbacks have been identified. Chief among these is the requirement that datagrams destined for the mobile unit be routed through the mobile unit's home agent [Sol98] [Per98]. This leads to a triangular routing pattern (mobile node to correspondent node to home agent to mobile node) and is referred to as triangle routing (triangle routing is discussed in detail in Section 4.4.1). Triangle routing can be particularly inefficient if the mobile node and the correspondent node are relatively close to each other, necessitating that packets deviate from an optimal route to pass through the home agent. A solution to this problem is to inform correspondent nodes of the mobile node's current care of address [Per98]. Such an association between the mobile node's address and care of address is known as a *binding* for the mobile node. Up to date bindings would enable the correspondent node to tunnel packets directly to the mobile node. However, this strategy would require binding updates to be transmitted from the mobile node to correspondent nodes. This raises a major security risk (since a malicious user could send a fake binding impersonating a mobile user and hijack their traffic) and security mechanisms to support this are still being worked out [Per98].

2.2.3 IPv6 Mobility Support

IPv6 (IP version 6) is the next generation of the Internet protocol (the current version is 4). Since IPv6 does not already have an installed base, it has been possible to make adjustments to the protocol to provide support for mobility [Per96b] [Per98]. Mobile IPv6 operates on the same basic principles as Mobile IP. Mobile nodes obtain a care of address on a foreign network, and inform a home agent about this care of address. The home agent is then able to tunnel datagrams to the care of address. However, Mobile IPv6 offers three major improvements over this base protocol.

The first improvement is that foreign agents have been eliminated altogether. Neighbor discovery [Nar96] and stateless address configuration [Tho96] provide a simple mechanism for obtaining a co-located care of address. Secondly, support for binding updates in correspondent nodes is incorporated as a part of Mobile IPv6 [Per96b]. All IPv6 nodes will be capable of caching the bindings (i.e. care of addresses) of mobile nodes with which they are communicating. Furthermore, IPv6 contains an ideal delivery mechanism for these binding updates, namely destination options. These are header options that are examined only at the destination node, as opposed to IPv4 header options, which are examined at each intermediate node and hence incur a performance penalty. In this way, the binding update may be included as a part of a regular IPv6 datagram transmitted to a correspondent node. The third improvement provided by Mobile IPv6 is that every IPv6 node will be able to establish and maintain security relationships with any other node as required [Per96b]. Hence authentication of binding updates will be possible through existing IPv6 security procedures. These last two points taken together solve the problem of triangle routing for Mobile IPv6.

2.2.4 Dynamic Service Discovery

Recently, the need for dynamic service discovery was recognized in mobile computing environments [Per96c, Hod99, Bre98, Bec99]. Inherent with mobility is the implication that the geographic location of the user is constantly changing. Recall that a mobile unit will generally interact with a number of different heterogeneous networks. The set of available network services and devices will depend on the current point of attachment and will likely be of interest to a mobile user. A common service used to illustrate this point is printing (i.e. locating and configuring printers on a foreign network). Some dynamic means of locating and configuring the desired services is required in these foreign environments. The following sections discuss the different protocols that have been developed to address this issue.

2.2.4.1 Resource Discovery Protocol (RDP)

The resource discovery protocol (RDP) [Per96c] is a lightweight protocol running over UDP that is used to dynamically discover network resources. Although useful in fixed computing environments, it is targeted to enable mobile users to find and access services external to the mobile unit. The requirements imposed on RDP were that the protocol be scalable, self managing, distributed, compatible with other administrative tools and compatible with mobile networking protocols. Uniform Resource Names (URNs) are used to identify the different types of resources in RDP (such as printers). The locations of these resources are specified using Uniform Resource Locators (URLs).

RDP employs a simple request/response mode of operation and functions in the following way [Per96c]. A local server (called the directory agent) is set up on a network. It is the responsibility of the directory agent to respond to all RDP queries. RDP supports dynamic registration and deregistration of services with the directory agent. This allows the directory agent to maintain a consistent set of URLs for all the currently available resources. Mobile units learn the address of a directory agent either through static configuration or dynamically through DHCP (via a DHCP configuration parameter). When a mobile unit wishes to access a particular resource, it queries the directory agent specifying the URN for the resource type and a number of keywords describing the resource (keywords provide values for resource attributes, such as the ability for a printer to use color, that allow the directory agent to select an appropriate resource). The directory agent responds with one or more URLs for resources that satisfy the query.

RDP was promising, but it was abandoned in favor of the service location protocol (SLP) discussed in the next section. However, a number of features of RDP were incorporated into SLP [Per96c].

2.2.4.2 Service Location Protocol (SLP)

SLP (defined in RFC 2608) [Gut99b] [Kem99b] is a protocol for dynamically providing hosts information regarding the existence, location and configuration of networked services. Perkins [Per96c] identified the potential benefits of employing SLP in a mobile computing environment. Currently SLP is in its second incarnation (SLP version 2) and the following discussion will revolve around this updated version.

SLP consists of three main components, which are described below [Gut99b]:

- User Agents (UA) – Processes that work on behalf of a user to obtain service information from service agents.
- Service Agents (SA) – Processes who work on behalf of one or more services, to advertise those services.
- Directory Agents (DA) – A process that acts as a cache for service advertisements. Although not strictly necessary, directory agents are used to provide scalability.

Similar to RDP, SLP employs a request/response model [Gut99b]. The basic operation of SLP consists of the following. A user agent, acting on behalf of a particular user process (client) issues a service request message specifying the characteristics of the service that the client requires. The user agent then receives one or more service replies indicating the location of network services matching the request. In cases where the user agent issues the service request directly to the service agents, the request is multicast on the network using the well-known SLP multicast address. Service agents receiving this request will respond with a unicast reply to the user agent if the service they advertise matches the service described in the request. Otherwise the request is ignored.

Directory agents may be introduced for reasons of scalability to prevent excessive multicasting on the network [Gut99b]. They function as a cache for service advertisements. Service agents register their services with the directory agent and user agents unicast service requests to the directory agent rather than multicasting them on the network. If a service has been registered with the directory agent that matches the service request, then a response is transmitted to the user agent. User agents may learn the location of a directory agent in a number of ways including static configuration, DHCP (RFC 2610 [Per99] defines a DHCP option used to specify the location of an SLP directory agent), passive directory agent advertisements (periodically the directory agent advertises its existence on the network) or by actively soliciting a directory agent advertisement (accomplished by sending a service request for a service of type directory-agent).

Services advertised through SLP are organized into groups called scopes [Gut99b], which are used for administrative purposes and access control. Every user agent is assigned a particular scope (either through static configuration or dynamically via DHCP). A user agent may only locate services that are within the same scope for which it is configured.

There are two components that describe every service advertised through SLP: service URLs and service templates [Gut99b, Gut99, Kem99b]. A unique service URL is associated with each service. The service URL provides information about the location of the service and how to access it (similar to URLs used by web browsers to locate and access web pages). A service type is included as a part of the service URL. Every advertised service is an instance of a particular service type. The service type specifies unambiguously how to use the service (and how to interpret the service URL). Service types may be a network protocol, an abstract service type or a concrete service type. Abstract service types indicate that the service may be associated with multiple protocols. Concrete service types specify a particular protocol within an abstract service type.

A service template is used to provide a formal definition of a service type, including the attributes that may be associated with the service (e.g. a printer service type may have an attribute that indicates whether the printer supports color). Specific instances of the template (i.e. services) set different attribute values according to the service they offer. These values are used to distinguish between different services of the same type. Service request messages may include values for these attributes to locate a service providing certain features. The grammars describing the construction of service URLs and service templates are contained in RFC 2609 [Gut99].

Example (adapted from [Kem99b]]): Consider the abstract service type "service:login" that defines a remote access service. Several hosts on the network may offer this service, including *castor* (a computer which supports remote access through the rlogin protocol) and *mira* (a computer which supports remote access via the telnet protocol). A service request for the abstract service type "service:login" would return the following service URLs:

- service:login:rlogin://castor.cc.umanitoba.ca
- service:login:telnet://mira.cc.umanitoba.ca

Each service URL specifies the location of the service (*castor* or *mira*) and the access protocol (rlogin or telnet). On the other hand, suppose a client only supports the telnet protocol. In this case, the user agent would query the service using the concrete service type "service:login:telnet" and only the second service URL (for *mira*) would be returned. (Note that in this example it is assumed that there exists service templates describing both the abstract service type "service:login" and the concrete service types "service:login:telnet" and "service:login:rlogin").

Beyond the functionality described above, SLP offers several other desirable features. It contains security mechanisms, providing for the authentication of both service URLs and their attributes [Gut99b]. Additionally, RFC 2614 [Kem99] defines a standardized API for SLP in both C and Java to promote portability among various implementations.

2.2.4.3 Other Service Discovery Models

In addition to the protocols discussed previously, several mobile computing frameworks employ their own techniques for dynamic service discovery. The MOCA service framework [Bec99] uses a distributed service discovery architecture, with no single repository of services (i.e. nothing comparable to the directory agent provided by SLP). Services are modeled as Java objects, which are downloaded by the mobile unit as they are required. Service advertisements are periodically multicast by the devices offering the services. It is unclear how this architecture would scale to support large numbers of services, since the purpose of a single service repository (e.g. the directory agent in SLP) was to provide for that scalability. A single repository would prevent the network from being overwhelmed by service advertisements or queries in the case where many services are being advertised.

The service discovery architecture presented in [Hod99] is similar to that of SLP. A service interaction client (SIC) is responsible for service discovery (like a user agent) and a service interaction proxy (SIP) handles service advertisements for a group of services (similar to a directory agent). However, this architecture is more focused on the discovery and usage of hardware type services, such as services to control the audio/visual equipment in a classroom. Additional functionality has been incorporated to download a description of the user interface for the service to the mobile client, so that the client can recreate a suitable interface and present it to the user.

2.3 Mobile Agents and Agent Systems

This section provides an introduction to mobile agents and agent systems including common terminology and basic mobile agent operation. The concepts described here are an integral part of the architecture presented in Chapter 3.

Mobile agents are an emerging technology that has evolved from the trend towards greater flexibility in network computing paradigms [Won99] [Lan98]. An agent is typically described as a program that is able to function autonomously, acting on a person's behalf to provide some service to that person. They operate within the bounds of an execution environment and have the following characteristics [Lan98]:

- Reactive – they are able to adapt to changes in their environment.
- Autonomous – they are able to operate independently, without direct control from a user
- Goal Driven – they work towards achieving a particular goal.

Stationary agents spend their entire lifetime executing at one location [Lan98]. Mobile agents are not bound to a particular system. They have the flexibility to transport themselves (or be transported by others) from one network to another. This flexibility engenders a new paradigm in network computing: the mobile agents paradigm. Several benefits can be realized from the usage of mobile agents, many of which are directly applicable to a mobile computing environment. Mobile agent benefits include (adapted from [Lan98]):

- Reduced network load and network latency. By transmitting a mobile agent to the location of a network resource (e.g. a database) and performing computations locally, they reduce the amount of network traffic generated when accessing the resource. Recall (Section 2.1.1) that wireless networks often experience high latency.
- Encapsulate proprietary protocols. By sending a mobile agent to the opposite endpoint of a communications link, a communications channel may be dynamically established to employ a proprietary protocol. Note that this benefit seems particularly relevant in a bandwidth limited environment such as those presented by wireless networks. A proprietary protocol tailored to a specific application has the potential to make better use of the available bandwidth.
- Mobile agents execute asynchronously and autonomously. They are capable of executing independent of user control. Again, recall from Section 2.1.1 that wireless networks experience frequent and unforeseeable disconnections. Agents that are able to operate autonomously may continue to perform computations on behalf of a mobile unit, even when it is disconnected.
- Mobile agents can adapt dynamically to changing conditions in their environment. Section 2.1.1 indicated adaptation is key to developing applications that run in a mobile computing environment

There are two fundamental components involved in the mobile agents paradigm: mobile agents and their agent systems. Mobile agents are software agents that are capable of changing the location at which they are executing. They may be characterized by five attributes [Lan98] [Obj97]:

- State – There are two types of state that can be associated with a mobile agent. The mobile agent's execution state and its object state. Execution state refers to the runtime state of the agent, including current values of the program counter and frame stack. Object state refers to the values of the internal attributes of the mobile agent. For instance, a piece of information the agent has retrieved from a database. Whenever a mobile agent is transmitted to a new location, its state must be transmitted as well. This enables the agent to resume execution where it left off, if desired.

- Implementation – Implementation refers to the actual code used to implement the mobile agent. When an agent is transmitted to a new location, its code must be transmitted as well. Generally there are two mechanisms by which this is accomplished: transfer all the code along with the agent when it is transferred, or transfer just the agent and load the code on demand, as it is needed.
- Interface – The mechanism exposed by the agent that allows other agents and agent systems to communicate with it.
- Identifier – A unique name used to identify the agent.
- Principals – The persons responsible for the actions of the agent. These include the manufacturer of the agent and the owner (person using the agent).

An agent system is a platform that is able to create, execute, transfer and terminate mobile agents [Lan98] [Obj97]. Mobile agents cannot execute outside of an agent system. Every agent system has an agent system type and identity associated with it. The agent system type is used to determine the kinds of mobile agents that the system supports. The identity is a name and a network address used to locate the agent system. An agent system resides on a particular host in a network. Within the agent system are a number of execution environments called places (see Figure 4.2). A place provides a specific context in which a mobile agent may execute [Obj97]. Together with the agent system, the place is able to offer security mechanisms such as controlling access to resources on the host. A place may contain many mobile agents and an agent system may contain multiple places. Each place within the agent system has its own unique address, which is a combination of the agent system name and the name of the place. Furthermore, a default place may be associated with the agent system, which is identified using the agent system identifier only. Similar to mobile agents, an agent system is also associated with a number of principals who bear legal responsibility for the system. Again these principals are the manufacturer of the system and the owner [Obj97].

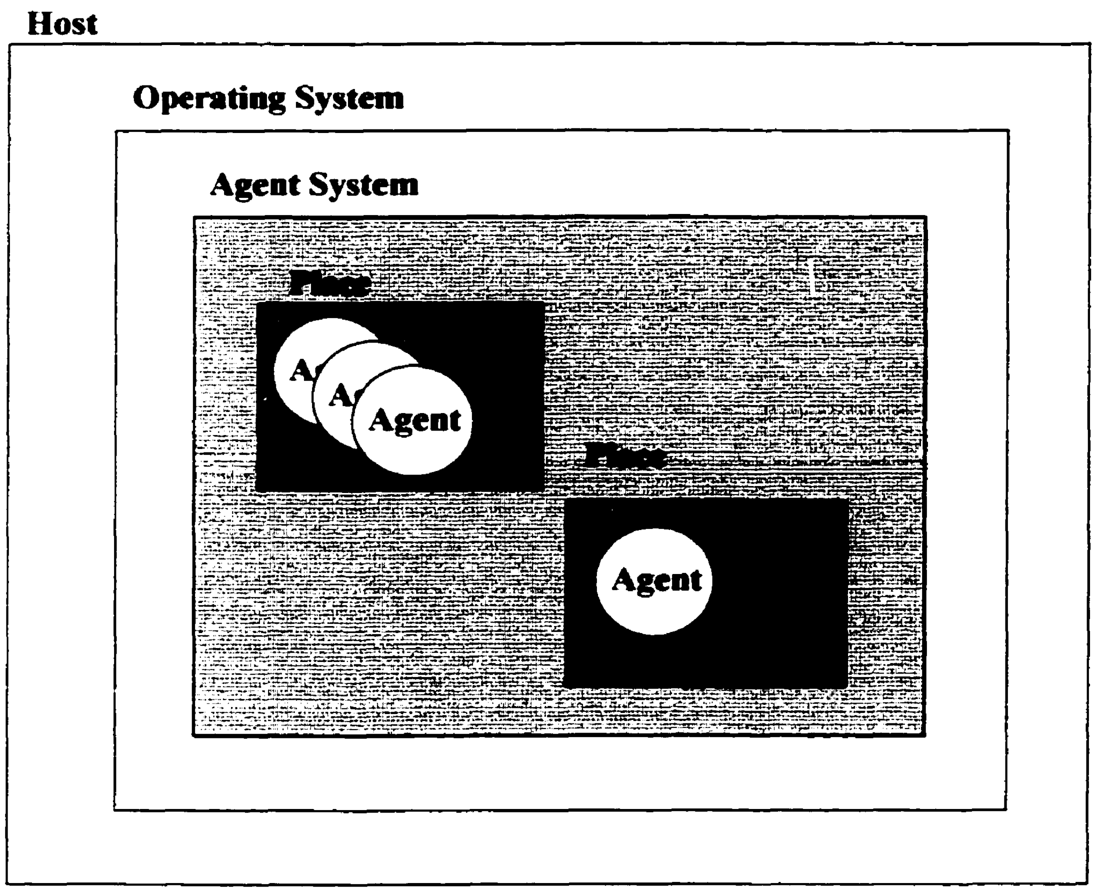


Figure 2.4 - Agent System Components

To foster interoperability among mobile agents and agents systems, standardization efforts are underway in the mobile agent field. In particular, the MASIF (Mobile Agent System Interoperability Facility) specification [Obj97] discusses interoperability between agent systems written in different languages and developed by different vendors. The specification addresses the interface between agent systems only and makes no attempt to standardize inter-agent communication or agent operations such as interpretation, serialization (storage of mobile agents), *etc.* MASIF standardizes the following four areas [Obj97]:

- Agent Management – Defines common ways to perform administrative functions such as creating, or terminating agents.
- Agent Transfer – Specifies ways to transfer agents between agent systems.

- Agent and Agent System Names – Allows agents and agent systems to identify each other and permits applications to identify specific agents and agent systems.
- Agent System Type and Location Syntax – Location syntax enables agent systems to locate one another and agent system type allows agent systems to recognize the type of agents that other agent systems support.

Generally mobile agent systems are implemented using interpreted or scripting languages for reasons of portability and security [Lan98]. To this end, Java has been identified as a promising language for mobile agent systems. Java offers multiplatform support and portability. Moreover, Java provides a number of features that are desirable for agent systems. Among these are [Won99, Tho97]:

- Support for object serialization – It is easy to convert a Java object and its associated object state to a form suitable for transmission across a network. Furthermore, it is simple for the receiving host to reconstruct the object.
- Support for code migration – Java's class loading mechanism supports dynamic retrieval and loading of Java class files (which contain the code used to implement Java objects)
- Support for security management – Java offers several security features. All classes loaded by the class loader are inspected carefully and subject to security constraints. Also, Java supports highly configurable security policies (e.g. the ability to control access to certain system resources, such as the filesystem)

A number of Java based mobile agent systems currently exist including IBM's Aglets [Osh97, Lan98] and Concordia [Won97].

Up to this point the chapter has focused on background material, reviewing the problem area in depth and providing an overview of the enabling technologies of mobile computing. The final section examines related work in the field of mobile computing that is relevant to the thesis.

2.4 Related Work

This section presents a selection of current research in mobile computing that directly relates to the architecture described in this thesis. Wireless access to the World Wide

Web has been a popular research topic for some time now. A number of systems [Baq95] [Cha97] [Hou96] [Jos99] [Lil95] [Sch96] [Voe94] have been developed to optimize web browsing in mobile computing environments. Typically these systems have relied on some form of proxy-based architecture to overcome the limitations imposed by wireless networks. Three systems have been developed that employ a client/intercept/server architecture as described in Section 2.4: Mowgli, WebExpress and Mobiscape.

Mowgli [Lil95] was one of the earliest mobile web browser systems to use the client/intercept/server architecture. It employed optimizations including prefetching of inline images, caching data on the mobile and fixed units, using persistent sockets to reduce connection setup and teardown times, and background fetching of pages (asynchronous operation). In addition, the Mowgli research identified promising optimizations that could be employed including web page compression, reduction of image quality, and annotation of HTML (to indicate to the user the size of the object pointed to by an HTML link and the availability of the object in the cache) [Lil95]. Other systems have since incorporated this functionality and demonstrated its usefulness [Baq95] [Jos99] [Sch96].

The WebExpress system [Cha97, Hou96] is very similar architecturally to Mowgli, but differs in both the goals of the system and the optimizations that it employs. WebExpress was designed to support transaction processing applications in a wireless environment and this goal is reflected in the optimizations that it uses [Hou96]. Typically, transaction processing (forms based) applications are characterized by repetitive and predictable responses. For example, submitting a request for a quote on a particular stock will return a standardized stock quote page with the desired information. Although the specific information contained in the quote changes with each request, the format and structure of the resulting page will remain the same. The optimizations of WebExpress take advantage of this fact. In particular, the key optimization performed by WebExpress, differencing, makes use of this information by storing templates of pages on the fixed and mobile proxies, and transmitting only the differences when new pages are requested. WebExpress also employs caching (on both the fixed and mobile unit), virtual sockets (persistent sockets used for many web page requests), header reduction and disconnected operation (by queuing requests while the mobile unit is disconnected) [Hou96].

Mobiscape [Baq95] is the third system to employ the client/intercept/server architecture. It focuses more on flexible caching policies that enable users to specify what information should always be available in the cache by way of a user profile. Furthermore, Mobiscape compresses HTML pages before they are transmitted across the wireless link [Baq95].

Several other systems have been implemented to support mobile web browsing that do not employ a client/intercept/server architecture. Mowser [Jos99] uses both a proxy based and end-to-end approach. The architects of Mowser argued that the end systems should be capable of providing different content to resource limited mobile devices. Hence Mowser attempts to perform content negotiation with the end systems in order to request data in a more suitable format (e.g. lower resolution images so that they do not waste as much bandwidth) [Jos99]. Additionally Mowser employs a wired side proxy capable of optimizing data before it is transmitted to the web browser (descriptions of specific optimizations are deferred until Section 4.5). The Teleweb system [Sch96] differs from other mobile web browser systems in that it focuses on overcoming the cost limitation of wireless networks. To achieve this goal, Teleweb performs a variety of functions including budget monitoring and annotating HTML to make cost information visible to the user. Teleweb is unique among the systems discussed here in that it employs a proxy that runs entirely on the mobile unit. All the other systems described so far rely on a wired side proxy of some sort. Similar to Teleweb, the Mobisaic system described in [Voe94] takes a unique approach to mobile web browsing. Mobisaic is concerned with dynamically adapting the information the browser displays with respect to the current location (context) of the mobile unit. It employs dynamic URLs that change (thereby causing the page referenced by the URL to change) as the mobile unit moves. This technique has the advantage that it allows a single URL to refer to multiple documents depending on the user's current environment [Voe94]. Hence a web page can provide context-based information by using dynamic URLs to link to other pages. However, Mobisaic does not address any of the limitations of wireless mobile computing environments and the functionality it provides is orthogonal to that offered by the other systems.

Other projects, although not specifically focused on mobile web browsing, have addressed the issue through their work. Zenel [Zen99] designed a dynamic proxy system based on the concept of downloadable "filters". These filters are executable binary programs that can be downloaded from the mobile unit (or other locations) and executed on the fixed network (in an execution environment) to offer services that overcome the limitations of wireless connections. The system is very general and appropriate for several different optimizations, both at the application and transport layers. Regarding web browsing, a HTTP filter was implemented as part of the test suite for the system [Zen99]. The filter is capable of compressing HTTP data before it is transmitted to the mobile unit. A counterpart process was required on the mobile unit to decompress the HTTP data as it arrived. The deployment of the HTTP filter (and the corresponding process on the mobile unit) represents a dynamic form of the client/intercept/server architecture, similar to the architecture described here. The comparison with our system will be examined in more detail in Sections 3.4 and 4.5.

Both the Dataman [Imi96] and BARWAN [Bre98] projects touched on the subject of mobile web browsing. The Dataman wireless web project [Imi96] dealt with improving the performance of popular Internet services when used over a wireless medium. It addressed issues both at the application and transport layer. Similar to the systems described earlier, the Dataman wireless web research focused on using a wired side proxy to manage the HTTP connections. The proxy is able to prefetch pages residing on the same web server as a previously requested page, and to change the format of pages (e.g. removing images) before they are transmitted to a mobile client if there is insufficient bandwidth. The Dataman research also suggested that the proxy itself could be mobile, moving along with the mobile user. This area was not yet explored in the project.

In the BARWAN project [Bre98] a general proxy model, TACC, was developed for use with applications executing on mobile clients. TACC is an acronym for transformation, aggregation, caching and customization, referring to the services provided by the proxy. Specifically, a proxy conforming to this model is capable of transforming the data before sending it to the web browser (applying lossy, datatype specific compression techniques to reduce the amount of data transmitted while retaining the semantic content), aggregating data from several sources and organizing it for the mobile user (a form of autonomous operation), caching both original and transformed copies of data and allowing users to customize the services the proxy provides through parameters. The TACC model was used to implement Top Gun Wingman, a web browser for PDAs.

The use of mobile agents in mobile computing environments has also gained attention in the mobile computing research community. Various papers [Jos99b, Jos97, Sah98, Gra96] have stressed the benefits that mobile agents offer to such environments, particularly with respect to support for disconnected operation (where a mobile agent can continue to operate on the fixed network while the mobile unit is disconnected). The Magenta mobile agent system [Sah98] was designed specifically for use in such weakly connected mobile computing environments. In addition to the usual mobile agent system functionality, Magenta offers a selection of mobile specific features. Magenta can adjust to the dynamic appearance and disappearance of execution environments and supports remote instantiation of mobile agents. A mobile client that is not running a copy of the Magenta agent system can launch a mobile agent on a remote site that will execute on its behalf. Magenta also provides tolerance of execution environment failures (designed to overcome abrupt disconnections or mobile unit crashes by maintaining backup copies of mobile agents) and a directory service for agents. The directory service allows mobile units to track the progress of agents they have dispatched when the mobile units reconnect following a period of disconnection. The focus of these optimizations is to overcome the instability of wireless connections, by allowing operation to continue while disconnected and upon reconnection of the mobile unit.

Likewise, Agent TCL [Gra96] is another mobile agent system that includes features to facilitate mobile computing and disconnected operation. Agent TCL offers several network sensing tools (including tools to determine connectivity and available bandwidth), which can be used for adapting the agent behavior. Furthermore, Agent TCL implements a docking system for disconnected operation. Each mobile unit has an associated docking station (a computer on the fixed network). The docking station is capable of storing agents destined for the mobile unit on disk while the mobile unit is disconnected and forwarding the agents to the mobile unit upon reconnection. The docking station alone is responsible for detecting connectivity of the mobile unit, and the process is transparent to the mobile agent (hence the mobile agent no longer has to handle disconnections).

Another related project, the Rover toolkit [Jos97, Jos97b], was created for developing mobile applications that support disconnected operation. The toolkit uses a distributed object system based on a client/server architecture with a client running on a mobile unit (typically) and a server running within the fixed network. The Rover architecture consists of two main entities: Relocatable Dynamic Objects (RDO) and Queued Remote Procedure Calls (QRPC). RDOs encapsulate code and data, and can be dynamically loaded from the client to the server or vice versa. They can also have their own thread of execution. QRPCs are used to invoke non-blocking RPCs on remote objects and to load RDOs from a server (or client). Although not strictly speaking a mobile agent system (RDOs are really just relocatable objects with no notion of execution state), Rover is able to support the construction of both mobile aware and mobile transparent applications. It supports disconnected operation (through the QRPCs), compression of headers and data before transmission and gives applications control over where the computation is performed. With respect to mobile web browsing, Rover was used to implement an HTTP proxy that provides compression of HTTP data, prefetching of inline images, and support for disconnected operation (by queuing web page requests).

Mobile agents have also been deployed in other mobile environments such as wireless cellular phone applications as described by La Porta [Por98]. In this project, the agents resided on the fixed network and were used to perform call setup negotiations on behalf of the phone (the mobile unit in this system). Mobility was required so that the agent could move inside the fixed network, thereby remaining close to the mobile user to reduce the call setup time.

Mobility frameworks identify services that should be provided by fixed networks supporting mobile clients. Hodes & Katz [Hod99] discuss one such framework. They outlined four technical requirements that must be provided by a fixed network that offers services to mobile clients:

- Device mobility
- Network accessible controllable objects
- Underlying service discovery architecture
- Mapping between the device interfaces and the interfaces presented to the mobile client.

Section 3.4 discusses how this framework is related to the architecture presented in Chapter 3.

Recall, Section 2.1.1 identifies adaptability as one of the key features of applications in mobile computing environments. Several projects have examined different ways of implementing this adaptability (including many of the mobile web browser systems mentioned earlier). Welling & Badrinath [Wel97] presented a general event framework by which mobile applications can receive notification of changing network resources (such as battery power on the mobile unit or bandwidth availability). This framework enables mobile applications to identify and adapt to changing network conditions. The Odyssey system [Nob95], an experimental Unix platform for mobility, applied the concept of data fidelity to mobile computing. Fidelity is defined as the degree to which a copy of the data presented to the user matches the original piece of data. Odyssey provides an API that allows mobile applications to make different tradeoffs among dimensions of fidelity (e.g. removing color from images) to increase the efficiency of the communications.

One area of related research not specifically linked to mobile computing, but important in the context of this thesis, is the work on active networks. Active networks employ mechanisms to place intelligence within the actual network [Cal98]. Their goal is to dynamically alter the behavior of the network as seen by the user. Active networks often accomplish this goal by injecting executable code into the network that can change the behavior of networking protocols from the application layer down to the networking layer (e.g. in the ANTS project [Wet98], mobility support similar to that performed by Mobile IP was dynamically added to a network that originally contained no such support). One branch of active network research that is particularly relevant with respect to this thesis is the active services approach (discussed by Amir et al. [Ami98]]). Active services restrict the user-defined computation to the application layer, thus allowing the benefits of an active network to be realized within the existing Internet infrastructure (without requiring existing Internet nodes be replaced by programmable counterparts). This restriction still enables many of the applications targeted by active networks research to be developed. The active service framework will be explored in greater depth in Section 3.4.

Chapter 3

SYSTEM ARCHITECTURE

This chapter introduces an architecture capable of supporting efficient web browsing (and, potentially, other networked applications) in mobile computing environments. The architecture describes the dynamic deployment of a client/intercept/server system within a mobile computing environment. Such a deployment transfers the benefits of the client/intercept/server architecture to mobile computing environments, while placing minimal architectural constraints upon those environments (refer to Section 3.2).

A foreign network is defined as any network a mobile user wishes to connect to that the user does not control (i.e. the user has no administrative privileges). Inherent in this definition is the idea that the user cannot install any proprietary software on the foreign network. Hence, architecture designed for such a network must function within the existing Internet infrastructure. The connectivity provided by foreign networks (as described here) is analogous to the network access offered by various Internet Service Providers (ISP). Each ISP provides a connection to a different local network (a foreign network on which the client has no administrative control), which is in turn connected to the Internet.

Note that this definition is somewhat more restrictive than the one presented in Section 2.2.2 (used in the Mobile IP standard). The reason for distinguishing a foreign network in this way is that, for a mobile application to be effective, it should work in any mobile computing environment the user may encounter (not only on one specific network with which the user commonly interacts). Such a requirement arises from the variability that is characteristic of mobility. Ubiquitous wireless network access will require users to interact with several different networks as they migrate from one locality to another. The restrictions regarding administrative control are added to ensure that the architecture presented here makes as few assumptions as possible about the foreign networks. Thus, it is possible that the mobile user may administer the foreign network, but it is not assumed to be true.

The mobile computing environment for this architecture appears in Figure 2.1. The mobile user is assumed to perform web browsing on a mobile unit, such as a laptop. The mobile unit services web page requests by communicating with a fixed network using a wireless network connection. The mobile computing environment is the combination of the fixed network (which provides some form of wireless network access) and the mobile unit. This is a typical mobile configuration. Note however that this description precludes *ad-hoc* wireless networks. The thesis architecture is designed to be effective in the environment described above. Finally, the foreign network is the fixed network part of the environment.

In the proposed architecture, there are two major components (see Figure 3.1): The mobile web browser support system and the mobility framework. These components interact to enable the optimization of application layer data across the wireless communications link in a mobile computing environment.

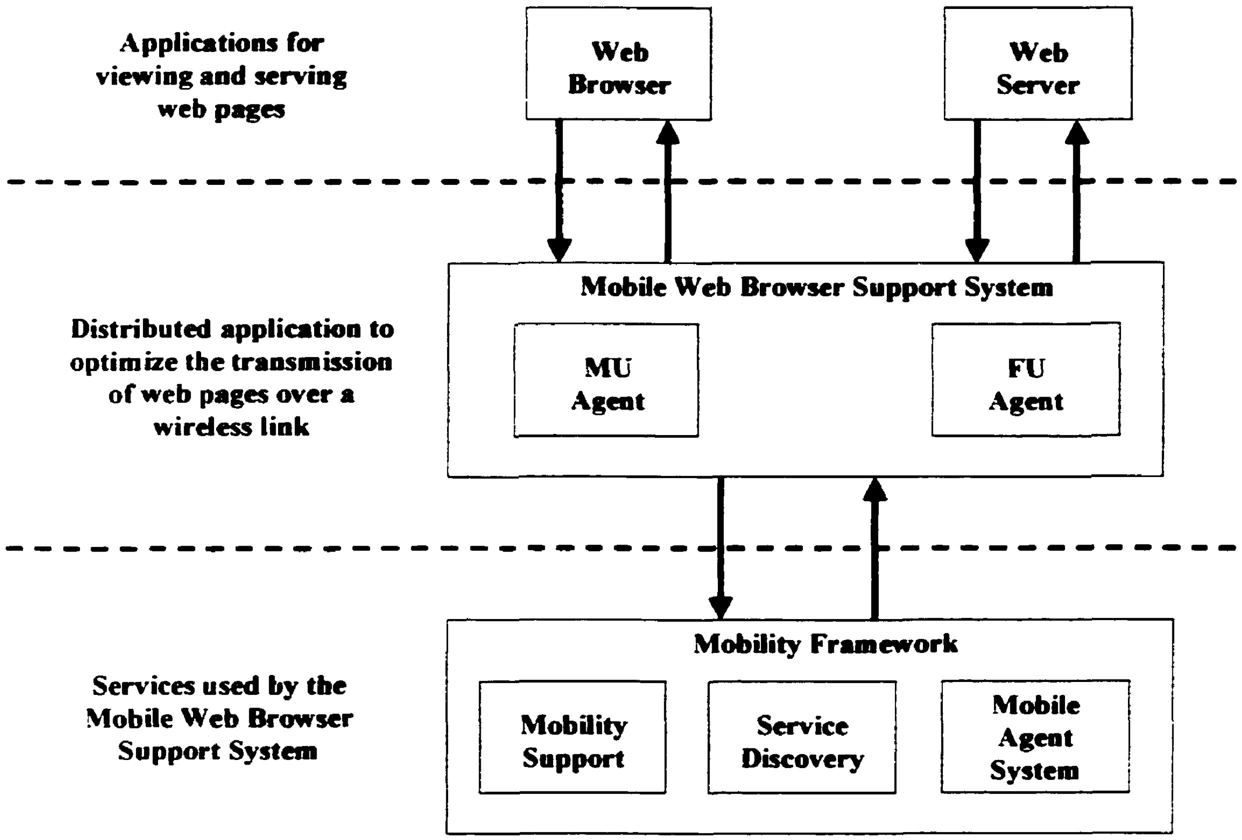

Applications for viewing and serving web pages
Web Browser
Web Server
Distributed application to optimize the transmission of web pages over a wireless link
Mobile Web Browser Support System
MU Agent
FU Agent
Services used by the Mobile Web Browser Support System
Mobility Framework
Mobility Support
Service Discovery
Mobile Agent System

Figure 3.1 - Software Architecture Model

The mobile web browser support system works on top of the mobility framework, utilizing the services that the framework provides (Figure 3.1). The support system operates at the application layer in a layered network model and employs mobile agents to dynamically deploy a client/intercept/server architecture in a mobile computing environment.

The mobility framework describes the basic set of services that are assumed to be available on a foreign network that is part of a mobile computing environment. Using a selection of the ideas presented in Chapter 2, the framework allows mobile units to make use of mobile agents in foreign networks where the location of network services are not known *a priori*. The framework includes support for mobility, dynamic service discovery and a mobile agent system. It is important to note that the framework does not rely on any proprietary protocols. The framework operates within the existing Internet architecture to facilitate access on a wide variety of networks. Although the framework must provide certain services, this thesis makes the assumption that mobile computing environments could offer these services in any event. If this assumption holds, then the framework does not place any new constraints on these environments.

The design goals of this architecture are twofold. First, the architecture must support the dynamic deployment of a system that enables operations to optimize application layer data transmitted across a wireless communications link. Second, the architecture must not require changes to established Internet protocols. The second requirement was particularly important due to the widespread deployment of the existing Internet infrastructure. Any architecture that imposes changes to this infrastructure for the purposes of optimization within a particular domain (here web browsing) is likely to encounter difficulties in the practical deployment of those changes. By satisfying the second requirement, the architecture will be effective on more foreign networks (subject to the assumption that the mobility framework is supported).

The following sections provide additional details on the architectural components with respect to the design goals (Section 3.1 and 3.2), a description of the interaction between the two components (Section 3.3) and a comparison with existing architectures for web browsing support in mobile computing environments (Section 3.4).

3.1 Mobile Web Browser Support System

At the heart of the architecture is the mobile web browser support system (Figure 3.2). The mobile web browser support system dynamically deploys a client/intercept/server architecture in a mobile computing environment to support optimizations that permit more efficient web browsing. The support system is composed of two agents (processes), called the FU agent and the MU agent. The FU agent is a mobile agent, capable of migrating from one agent system to another. The FU agent is dispatched into a foreign network to handle the server side processing of the client/intercept/server architecture. It operates on the server side of the wireless link connecting the mobile unit to the foreign network. The MU agent is always resident on the mobile unit. It handles the client side processing in the client/intercept/server architecture and operates on the mobile side of the wireless communications link.

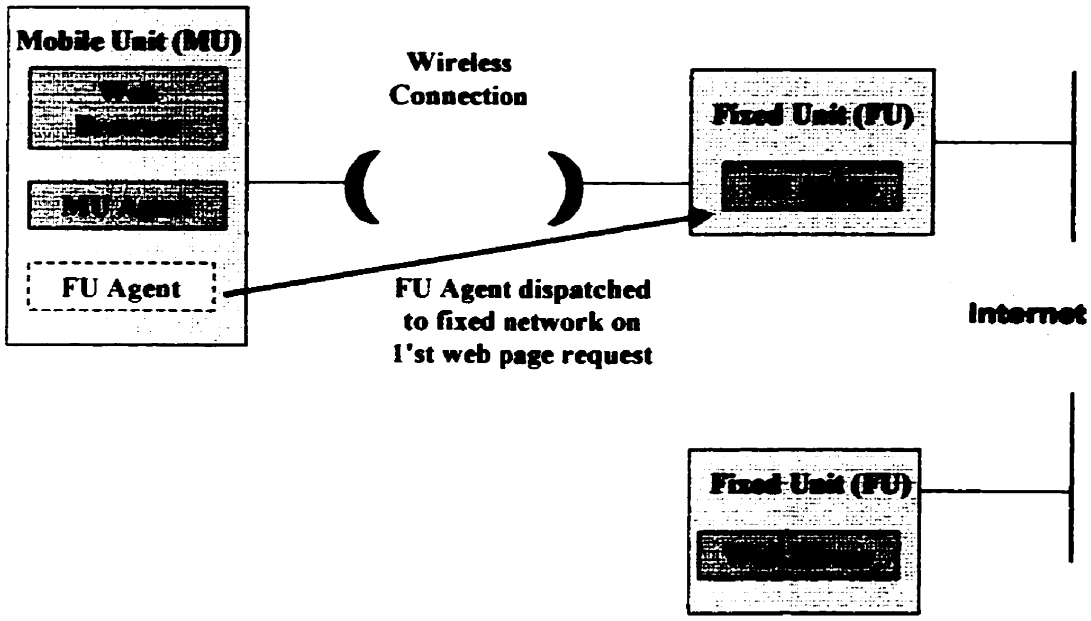


Figure 3.2 - Mobile Web Browser Support System

The MU agent is also responsible for handling dispatch of the FU agent, accepting requests from a web browser and communicating those requests to the FU agent. The FU agent is responsible for servicing requests and returning the requested entities to the MU agent. By dispatching a FU agent into the foreign network, the MU agent is able to achieve the benefits of a client/intercept/server architecture without assuming *a priori* that a FU agent is resident on the foreign network (an assumption that will often prove false). An assumption is made that the mobility framework presented in Section 3.2 is supported. Once a FU agent is dispatched into the foreign network, the MU and FU agents can completely control the format and type of the application layer information flowing across the wireless portion of the communications link. They can perform several operations to attempt to increase the efficiency of the application layer transmissions. The key feature of this architecture is the mobility of the FU agent.

The mobile web browser support system satisfies the design requirement of dynamically deploying a system that implements the optimizing operations. As demonstrated by Liljeberg *et al.* [Lil95] and Housel & Lindquist [Hou96] a wide variety of optimizations are possible once a client/intercept/server architecture has been established. The specific optimizations employed are implementation dependent and not addressed by the support system architecture. They may be tailored for the type of web browsing targeted by the system designer (e.g. different optimizations may be used when supporting *ad-hoc* web browsing vs. transaction based web browsing).

3.2 Mobility Framework

The mobility framework operates beneath the mobile web browser support system. It describes a collection of services that a foreign network in a mobile computing environment must provide to the support system. In order for a mobile agent (the FU agent) to execute on a foreign network, there are 3 services that the foreign network must offer: mobility support, mobile agent system and agent system discovery. Thus, the mobile unit must have a means of connecting to the foreign network. The foreign network must provide an execution environment for mobile agents (an agent system) and there must be a dynamic means by which the mobile unit may locate this execution environment. Dynamic location of an agent system is especially important because the mobile unit may have no advance knowledge of the foreign network with which it is communicating. Within this framework, the agent system in which the FU agent executes is presented as just another service that the foreign network makes available to mobile units (similar to other services the mobile user would need to access, such as a printer).

The mobility framework satisfies the second design goal in that none of these services requires any changes to existing Internet protocols (see Section 4.2). The framework does not specify how these services are to be implemented, only that they must exist on a foreign network as the architecture relies on them to connect to the network and to deploy the FU agent.

It should be noted that the security aspect of the framework is not specified. Security is extremely important in an architecture such as this because unauthorized users must be denied access to the services provided by the mobile computing environment. Each of the services mentioned above has certain security requirements. It is the responsibility of the providers of these services to ensure that the services themselves are secure. Indeed, many of the protocols that are used to implement these services contain mechanisms for secure operation.

3.3 Interaction of Components

This section describes the interaction of the mobile web browser support system and the mobility framework using a simple scenario.

At the highest level, an implementation of this architecture would operate in the following way. A mobile unit on which the mobile web browser support system is installed encounters a foreign network (e.g. user migrates to a new location that contains a wireless network connection) that supports the mobility framework. The unit connects to the network using the mobility service provided by the mobility framework. A user starts the MU agent on the mobile unit and then initiates a web browsing session. Requests from the web browser are passed to the MU agent, which first checks if it has already dispatched a FU agent to the foreign network. If it has, the request is forwarded to the FU agent and MU agent processing halts. If a FU agent has not been dispatched, the MU agent attempts to discover the location of an execution environment for mobile agents on the foreign network, using the service location facilities provided by the mobility framework. In the event that no agent system is discovered on the foreign network, the MU agent may either use a direct connection to satisfy the request, or indicate to the user that the request cannot be satisfied (this decision is implementation dependent).

If an agent system is discovered on the foreign network, the MU agent dispatches the FU agent to the agent system. The FU agent sends a handshake message to the MU agent once it arrives at the agent system and begins execution. At this point, the system has evolved into a client/intercept/server architecture, similar to those described earlier [Baq95, Hou96, Lil95]. The operation of the system proceeds as any client/intercept/server architecture would, whereby the MU agent forwards requests from the web browser to the FU agent for processing. The FU agent handles the request, contacting a web server to retrieve the requested entity and transmitting the entity back to the MU agent after the entity has been retrieved. Once the MU agent receives the entity from the FU agent, it forwards the entity to the browser. Since the FU and MU agents have complete control over the format and type of information that flows across the wireless link at the application layer, they are able to perform various operations to try and optimize the efficiency of the transmissions.

This scenario demonstrates how the services provided by the mobility framework are used within the architecture to set up a proprietary communications channel across the wireless link from the mobile unit to the foreign network via the MU agent and the FU agent.

3.4 Comparison with Related Architectures

Before comparing this architecture to others, it is helpful to identify a taxonomy of existing architectures, and to classify the other systems within this taxonomy. In general, the architectures presented in Section 2.4 to optimize web browsing within mobile computing environments share a common feature; they make use of a proxy to assist their optimizations. The number and location of the proxies can be used to classify these architectures. The taxonomy consists of the following classifications: Single proxy architectures where the proxy resides on the mobile unit, single proxy architectures where the proxy resides on the fixed unit, two proxy architectures which place a proxy on both the mobile and fixed units, and dynamic two proxy architectures.

Single proxy architectures where the proxy resides on the mobile unit (Figure 3.3) are the least flexible of all. An example of such an architecture is presented by Schilit *et al.* [Sch96]. These architectures are limited in the optimizations they may employ, since they have no counterpart operating on the fixed network. It is impossible for them to actually change the fidelity of data transmitted across the wireless link or to perform any optimizations on the data itself. This type of architecture cannot directly overcome the problems presented by wireless networks. It is limited to optimizations concerning disconnected operation, cost monitoring and so forth. These optimizations can only prevent the user from performing costly operations. They can do nothing to alleviate the cost itself. Thus, they avoid the problems presented by wireless networks rather than solving them. In cases where this avoidance is undesirable or unacceptable, the problems will still be present and no solution will exist. The biggest advantage of this architecture is that it places no reliance on the existence of a wired side component; hence it can be used effectively in any mobile computing environment.

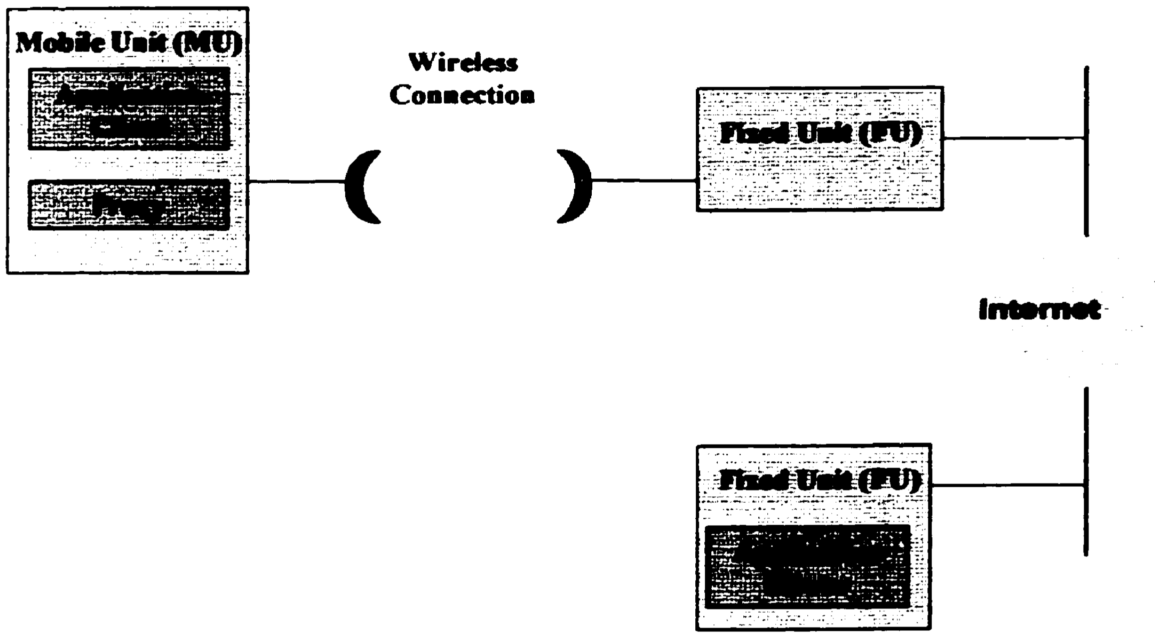


Figure 3.3 - Single Proxy Architecture, Proxy on Mobile Unit

Single proxy architectures where the proxy is resident on a fixed unit (Figure 3.4) are capable of performing application layer optimizations that overcome some of the limitations of wireless networks. An example of this type of architecture is Mowser [Jos99]. Since the proxy is resident on the fixed network, it is able to change the fidelity of data before it is transmitted to the mobile client. Possible optimizations include changing the color and/or size of images, dropping frames from movies or disconnected operation. However the optimizations will be one way only. Data flowing from the mobile unit to the fixed unit will not be optimized in any fashion. Furthermore, certain optimizations that require a counterpart executing on the mobile unit will not be possible. The most notable optimization with this requirement is compression, which necessitates a corresponding proxy on the mobile unit to uncompress any compressed data from the fixed unit. The main disadvantage present in this architecture is that the mobile unit must rely on the existence of a proxy in the fixed network with which it is communicating. Since the proxy is a proprietary piece of software, the assumption that it will exist on any given network is not valid. This is a problem because the user is mobile and could interact with any number of different networks, some of which may have the proxy installed and others that may not. So the optimizations will only be realized on certain networks, unless the mobile unit is able to access the proxy remotely. Further, remote access introduces additional problems of scalability (a single proxy used for potentially many mobile users) and latency (all data passes through the proxy, which is not located on the local network providing connectivity to the mobile unit).

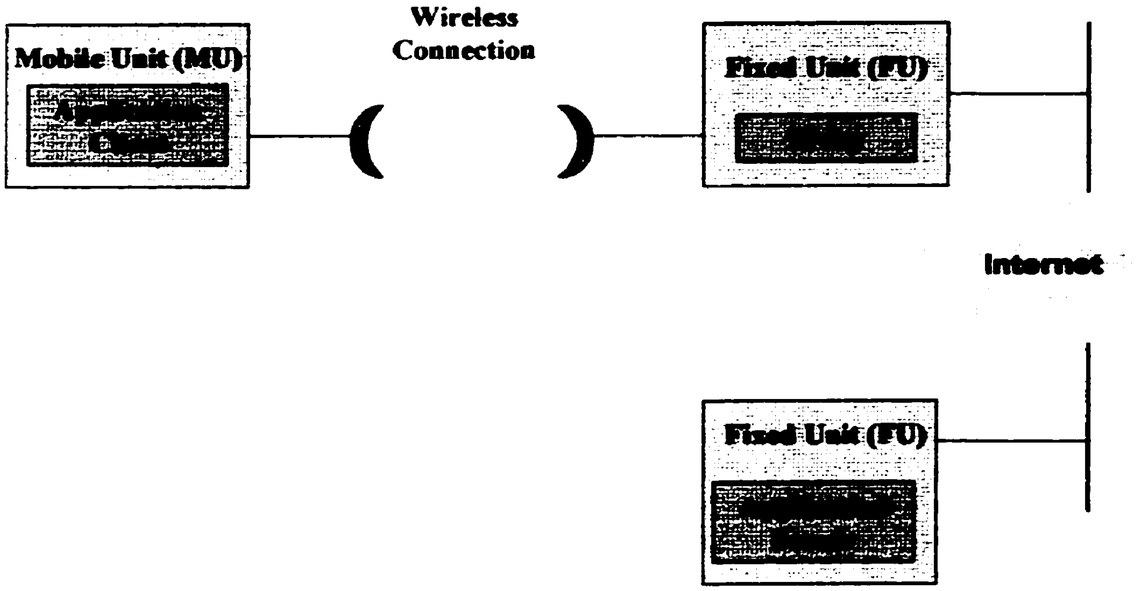


Figure 3.4 - Single Proxy Architecture, Proxy on Fixed Unit

Two proxy architectures (also referred to as client/intercept/server architectures, recall Figure 2.3) offer more flexible optimizations than either of the single proxy architectures. Since a proxy is resident on both the mobile unit and a fixed unit, these architectures are able to support any of the optimizations possible in either of the previous two architectures. To implement an optimization from a single proxy architecture using a two proxy architecture, simply implement the optimization in the appropriate proxy (on the fixed or mobile unit) and ignore the second proxy. Additionally, two proxy architectures enable more complex optimizations, such as differencing or compression, which require a proxy at either end of the wireless link. Similar to the single proxy architecture that places the proxy on the fixed unit, this architecture suffers from a reliance on a wired side component, thereby limiting the potential for optimizations when the user moves across various networks.

The final architecture in the taxonomy is the one described in this thesis, a dynamic two proxy architecture (Figure 3.2). In this case, the proxy on the fixed unit is actually dispatched by the mobile unit. Hence, the two proxy architecture may be dynamically deployed on any mobile computing environment that supports the mobility framework. The main drawback of the two proxy architecture, a reliance on a wired side proxy, is replaced by a reliance on a framework that can be implemented with existing Internet protocols. Furthermore, the benefits from a two proxy architecture can still be realized. The system presented by Zenel [Zen99] also made use of a dynamic proxy architecture but did not take into account the necessary mobility framework for deploying a proxy onto a fixed network. Rather than working within the existing Internet architecture, his system [Zen99] required changes be made to existing protocols (most notably Mobile IP) to be effective. Such a reliance on proprietary protocols yields the same problems as a reliance on a wired side component in that it limits the mobility to networks that support those proprietary protocols.

It is interesting to note the similarity between the architecture presented here and the active networks architecture of Calvert *et al.* [Cal98]. Effectively the MU Agent is injecting a form of intelligence (via the FU Agent) into the network to change the behavior of the network across the wireless communications link (by optimizing the transmissions). However the active network architecture is primarily concerned with optimizing network layer protocols (e.g. dynamically adding new protocols to a network) and routing by changing the behavior of network routers. It is not as suitable for performing optimizations at the application layer (e.g. optimizing HTTP, an application layer protocol) because routers do not have access to the application layer data. Routers work with packets that encapsulate the application data. Furthermore, the active network approach "...proposes that the Internet service model be replaced with a new architecture in which the network as a whole becomes a fully programmable computational environment" [Ami98]. Conversely, the architecture described in this thesis fits within the existing Internet infrastructure.

There is also a degree of similarity between this architecture and the active service framework discussed by Amir *et al.* [Ami98]. The active service framework has identified the need for services similar to those provided by the mobility framework. In particular, their service environment corresponds to an agent system and their service location mechanism corresponds to service discovery. They have no dual to the mobility service (providing connectivity to mobile nodes), since the active service framework was not designed for use in mobile computing environments. Moreover, the active service framework effectively deploys a single proxy into a network, since it makes no provisions for a corresponding proxy operating on the client. Hence the optimizations that can be achieved are limited to those possible in a single proxy architecture. It should be noted though, that the active service framework architecture does support composition of services, which could potentially offer other optimizations. However, this feature is still under development [Ami98].

It is likewise interesting to compare the mobility framework described here to the technical requirements for fixed networks outlined by Hodes & Katz [Hod99]. Specifically [Hod99] requires fixed networks provide support for device mobility, network accessible controllable objects, underlying service discovery architecture, and mapping between exported and device controlled interfaces. The first three requirements can be mapped directly to the mobility framework. Device mobility is identical to the mobility support requirement of the framework. Network accessible controllable objects can be generalized to a set of services that the network provides. The agent system in the framework is then just an instance of one such service. Furthermore, to facilitate agent system discovery, the mobility framework also relies on an underlying service discovery architecture provided by the network. There is no dual to the mapping of interfaces, as this requirement is more specific to the architecture presented by Hodes & Katz [Hod99], which focuses on the remote control of hardware device type services. However the basic requirements imposed by both frameworks can be viewed as equivalent, where Hodes & Katz [Hod99] is a more general case of the specific requirements imposed by the mobility framework.

3.5 Advantages

Several benefits of this thesis' architecture are noteworthy. The wired side proxy in this architecture (the FU agent) no longer represents a single point of failure for the system because another proxy can be dispatched into the network in the event that the first proxy fails. The proxy will always be close to the mobile unit (because it is dispatched into the network with which the mobile unit is communicating), thereby providing a more direct route for data rather than using a proxy installed on a remote network. Furthermore, flexible deployment of the proxy is permitted. In those instances where a FU agent (proxy) is not required on the fixed network (e.g. the mobile unit is docked to a fixed network and communicating via an Ethernet interface), the MU agent can simply choose to not dispatch the FU agent and employ regular HTTP connections to satisfy the web page requests. Only a slight performance penalty will be paid, resulting from the indirection of passing requests through the MU agent instead of directly to the web server. The reliance of other systems on a wired side component begs the question: is the proxy always accessible? What if the network the user is accessing is in isolation (i.e. a corporate intranet not connected to the larger Internet), or protected by a firewall? Some firewalls will block a proprietary protocol connection to a proxy outside the firewall (since some firewalls are configured to support only certain protocols, HTTP for example, and connections using unsupported protocols are blocked) [Cha95]. By dispatching the proxy into the fixed network, the proprietary channel is established inside the firewall and all communications through the firewall rely on standard protocols that are likely to be allowed to pass (e.g. HTTP). A dedicated wired side proxy that resides on a fixed host somewhere within the network is not flexible enough to handle these situations. Finally, the architecture presented here permits implementation with existing Internet protocols. Although the end system approach advocated by other systems (e.g. Mowser [Jos99] suggests end systems can provide mobile users with different content than fixed users) is certainly effective, it is difficult to deploy such a system within the Internet since there is a significant installed base of web servers that would have to be changed. Given the volume of existing servers, such a scheme is not necessarily practical.

Chapter 4

SYSTEM IMPLEMENTATION

This chapter describes the implementation of a prototype mobile web browser support system based on the architecture discussed in Chapter 3. The prototype was developed to assess the feasibility of the architecture. A performance analysis is not included as such an analysis would be reflective of the performance of the specific operations employed for optimization. This thesis presents no new optimizations, but instead a mechanism by which to deploy a system the permits optimizing operations at the application layer. The prototype was developed to assess the behavior of this architecture, not the performance. Each of the two architectural components presented earlier is examined again with respect to the specific implementation of the prototype (Sections 4.2 and 4.3). Implementation issues arising from the architecture are also discussed (Section 4.4) and the prototype is compared with implementations of similar systems (Section 4.5). The prototype operation proceeds exactly as outlined in Section 3.3.

4.1 System Hardware

The prototype system was designed for operation in IP networks only. It was implemented on a Windows NT network (see Figure 4.1) consisting of one fixed unit and one mobile unit. The fixed unit (*peppertree*) is a typical desktop computer with a Pentium (266 Mhz) processor and 64 Megs RAM running Windows NT. The mobile unit (*mosquito*) is an IBM ThinkPad (133Mhz) laptop with 32 megs of RAM running Windows 95. Communications between *peppertree* and *mosquito* takes place using Lucent WaveLAN wireless network cards (one is installed in both *peppertree* and *mosquito*) via radio waves operating at 2.4 GHz. WaveLAN supports data transfer rates of up to around 2 Mbps and ranges of approximately 250 meters [Luc98]. A TCP/IP protocol stack is used on top of the wireless interface for addressing and data transmission. *Peppertree* has a statically configured IP address and is set up to act as a DHCP server for a local IP subnet. *Mosquito* obtains its IP address dynamically through *peppertree* using DHCP (for more information on this, see Mobility support Section 4.2.1).

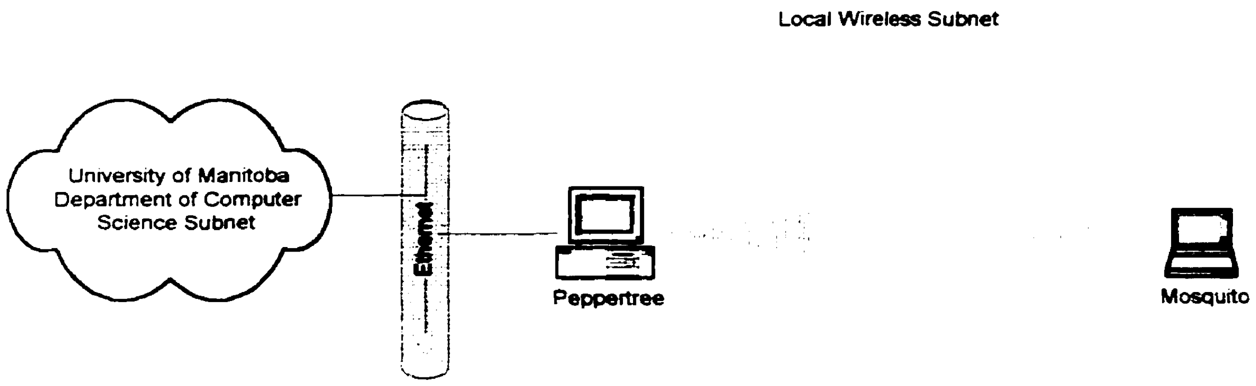

Local Wireless Subnet
University of Manitoba Department of Computer Science Subnet
Ethernet
Peppertree
Mosquito

Figure 4.1 - Wireless Network Testbed

4.2 Mobility Framework

As discussed in Section 3.2, the mobility framework refers to the services that must be provided by a foreign network to support an implementation of the architecture described in this thesis. These services include support for mobility, a mobile agent system, and some means of agent system discovery. The following sections describe how the services comprising the mobility framework were implemented on the test network to support the prototype mobile web browser. It is important to note that each of these services was implemented using existing Internet protocols. Hence, the architecture proposed in Chapter 3 does not require any extensions to the existing Internet protocols.

4.2.1 Mobility Support

Mobility support refers to the ability of a foreign network to offer network connectivity to visiting nodes. In the case of the test network described in Section 4.1, this amounts to requiring that the fixed unit (*peppertree*) provide visiting mobile units (*mosquito*) with a valid IP address for the subnet in which the fixed unit is configured. There are two options (recall Section 2.2) for doing this: Mobile IP and DHCP. DHCP is selected because it is simpler to set up and configure. Furthermore, DHCP can be installed within the confines of the local subnet used by the test network. Mobile IP requires that another subnet be involved (since the mobile unit would require a home network in addition to the foreign network that the test network provides). For the purposes of the architecture presented here either solution is acceptable. DHCP is chosen because it is the simplest and most readily available.

DHCP is provided on the test network by installing the Microsoft DHCP server on *peppertree*. The server was configured to respond to any DHCP requests that arrived on the wireless network interface (i.e. requests that come from mobile units wishing to connect to the network) offering IP addresses on the local subnet. *Mosquito* (the mobile unit) was configured to obtain an IP address dynamically using DHCP (via the windows DHCP client). When *mosquito* boots up within range of *peppertree*, it obtains a valid IP address that allows it to communicate with the network. This simulates the entry of a mobile unit into a foreign network. The mobile unit obtains an IP address so that it may communicate with the foreign network but has no other *a priori* knowledge of the network

The main drawback of this approach is security (which is a problem with DHCP in general [Dro93]). To provide network access to any visiting mobile units, the DHCP server was configured to service any DHCP request that it received from the wireless subnet (i.e. any DHCP request that arrives on the wireless Network Interface Card). This is a major security hazard, since any mobile unit with a wireless NIC and a DHCP client installed could access the network, even if the mobile unit is not authorized to do so. DHCP provides no easy solutions to this short of configuring the DHCP server with the MAC addresses of all mobile units that could potentially communicate with the server. Such a solution is acceptable, but difficult to administer since all legal MAC addresses must be accounted for [Per95]. This prototype implementation did not deal with the security issues arising here.

4.2.2 Mobile Agent System

The mobile agent system is responsible for providing an execution environment on the foreign network for the mobile FU agent. The type of mobile agent system chosen places constraints on the FU agent itself, which must be implemented as an agent that is supported by the agent system type. For example, if the FU agent is implemented as an aglet (a mobile agent in IBM's Aglets system) then the FU agent may only execute within an aglets server (the Aglets execution environment). The FU agent will not be compatible with other mobile agent systems. Although mobile agent systems are moving towards interoperability [Obj97], it has not yet been realized. Hence, for the purposes of this prototype, the FU agent must be a particular agent type and will only work on an agent system corresponding to that type.

In this prototype implementation, the mobile agent system employed was IBM's Aglets system. The Aglets system is similar to other mobile agent systems described in Section 2.3. It was selected because it provided the features necessary to effectively implement the prototype. The Aglets system is implemented in Java. Java was a requirement, since the mobility framework must be portable to any foreign network and Java is widely supported on many different platforms. The API provided by the Aglets development environment was also well suited for this prototype. It allows a mobile agent to be dispatched from a computer that does not host an agent system of its own. It must be possible to dispatch the FU agent from a mobile unit that is not running its own copy of the execution environment (as such a copy would be an unnecessary burden on the resources of the mobile unit). Often, mobile agents are dispatched from one agent system (execution environment) to another, using the functionality provided by the agent system to perform the actual dispatch operation. This approach was not an acceptable solution for this implementation because of resource constraints imposed by the mobile unit. As such, the agent system employed must provide a mechanism to dispatch a mobile agent to an execution environment on a foreign network from a host that is not running a similar environment. The Aglets agent system provides such a mechanism through its client API.

The Aglets system contains a development kit (including an API and the necessary implementation framework) for constructing aglets (mobile agents in the Aglets system) and a ready-made aglets server (agent system) called Tahiti [Ibm98]. Tahiti is an application program that provides a place for aglets to execute. It also offers aglet management services (e.g. the ability to create new aglets, *etc*) as well as the ability to grant aglets access privileges for various resources on the host on which it is executing.

In the prototype, the FU agent was implemented as an aglet, so that it would be executable within an aglets server. The Tahiti server was installed on *peppertree* (the fixed unit in the test network) as a part of the mobility framework, to satisfy the agent system requirement. Using the Aglets client API, the MU agent is able to dispatch the FU agent to the Tahiti server, where it may execute as a Java program with restricted access to local resources (similar to an applet in a web browser).

4.2.3 Agent System Discovery

Agent system discovery refers to the method by which the MU agent is able to determine the location of the execution environment for the FU agent (here the Tahiti Aglets server) on a foreign network. Using DHCP, the mobile unit is able to connect to the foreign network but the mobile unit has no knowledge about the location of services provided by the network. Specifically, the MU agent on the mobile unit does not know the location of the mobile agent system (if one even exists). The agent system discovery mechanism adopted for this prototype implementation was SLP. SLP (see Section 2.2.4.2) supports dynamic service discovery on a network where a node does not know the network configuration in advance (e.g. SLP may be used to determine the location of a printer). By modeling the agent system as just another service that a mobile unit could potentially access, agent system discovery arises as a natural consequence of the operation of SLP. This makes SLP well suited to the task of agent system discovery.

Each service advertised by SLP has a service: URL that is defined using a service template (see RFC 2609 [Gut99] for information on service templates). These templates specify the attributes of the service and the syntax for describing the location of the service. To model a mobile agent system as a service, a service template must be constructed describing the service. To this end, two different service templates defining a mobile agent system service were drafted (see Appendix A for the actual templates). The first template defines an abstract agent system service type called *x-agentsystem*. Abstract service types are generic in the sense that they define a type of service (here agent system) but do not specify which protocol to use when accessing the service. An abstract service type was used initially because there are many different agent systems available and each could (and most likely does) use a different communications protocol. The "x" in the name indicates that the service type is experimental. The second service template describes a concrete service type for the aglets agent system (actually, all agent systems using the agent transfer protocol) called *x-agentsystem:atp*. This template indicates how to access an agent system service using the agent transfer protocol (ATP). It defines the location syntax for agent systems employing the ATP protocol for agent transfer and communication. The service: URLs used in the prototype to indicate the location of the Tahiti aglets server are based on the *x-agentsystem:atp* template. Both service types (*x-agentsystem* and *x-agentsystem:atp*) were defined under a local naming authority, indicating that they are local to this particular implementation (since they are not standardized). Furthermore, these templates are minimal in the sense that they do not specify any additional attributes for agent systems other than location syntax (which was required for this prototype). The templates could (and should) specify a number of attributes that could be used when determining whether a particular agent system is suitable or not. For example, attributes could be used to query the access privileges granted to mobile agents by a particular agent system. A mobile agent may not wish to be dispatched if the destination agent system does not offer certain privileges that are required by the agent.

In addition to service templates for an agent system, an implementation of SLP was also required. For the prototype, a minimal implementation of SLP was constructed using Java. The implementation provides no support for directory agents or service requests that query attributes of a service. Only basic service advertisement through service agents and service discovery via user agents is supported. The implementation consists of two components: the SLP framework and the SLP API. The SLP framework provides the actual implementation of the SLP protocol, as specified in RFC 2608 [Gut99b]. It is capable of both advertising services (providing service agent functionality) and querying services (providing user agent functionality). The SLP framework is accessed through the SLP API. The API implemented here corresponds to the standardized Java API for SLP defined in RFC 2614 [Kem99]. The purpose of the API is to abstract the details of the SLP framework implementation. The API isolates the design decisions of this SLP implementation from the applications accessing the implementation. In this manner, the implementation of SLP may be changed without requiring changes to applications using SLP since the applications access SLP through the standardized API.

To provide agent system discovery (part of the mobility framework) on the foreign network using the SLP implementation described above, two additional components were constructed:

- A service agent to advertise the agent system service on the foreign network.
- A user agent to query the location of the agent system.

The service agent is an independent application process that uses the SLP framework via the SLP API to advertise the Tahiti agent system. The service agent was installed on *peppertree* so that the mobile unit would be able to determine the location of the agent system. The user agent functionality was incorporated into the MU agent, which uses the SLP framework to determine the location of the agent system before dispatching the FU agent. It is important to note that both the service agent and MU agent (user agent) only access the SLP framework through the SLP API. The SLP framework implemented for this prototype is not complete (it is only a minimal implementation). By using the SLP API, it is a simple matter to replace the SLP framework used here with a complete implementation if one becomes available, without requiring any changes to the MU agent or agent system service agent.

The interaction of the components used to provide service discovery in this prototype is shown in Figure 4.2. The service agent, executing on *peppertree*, uses the SLP API to register a service: URL (based on the x-agentsystem:atp template) describing the location of the Tahiti agent system with the SLP framework. The framework will now advertise the service on behalf of the service agent. Specifically, it will respond to any

SLP queries for services of either type x-agentsystem or x-agentsystem:atp with the service: URL registered by the service agent (indicating the location of the Tahiti server). The MU agent uses the API to query the location of a service of type x-agentsystem:atp (which is the type of agent system required to host the FU agent) through the SLP framework. The framework multicasts service request packets on the local subnet requesting a service of type x-agentsystem:atp. When the framework executing on *peppertree* receives such a service request, it will respond with a service reply packet containing the service: URL (indicating the location of the Tahiti aglets server) registered by the agent system service agent. This service reply will be received by the framework on *mosquito* and returned to the MU agent. The MU agent can extract location information from the service: URL contained in the service reply message (using additional methods provided by the API) and dispatch the FU agent to the Tahiti aglets server on *peppertree*. In the event that no service agent is running on *peppertree* to advertise the agent system service, the SLP framework on *peppertree* will not respond to SLP queries from the mobile unit requesting a service of type x-agentsystem:atp. Eventually the framework on *mosquito* will time out and will inform the MU agent that no compatible agent system is available on the foreign network. In this case the FU agent will not be dispatched.

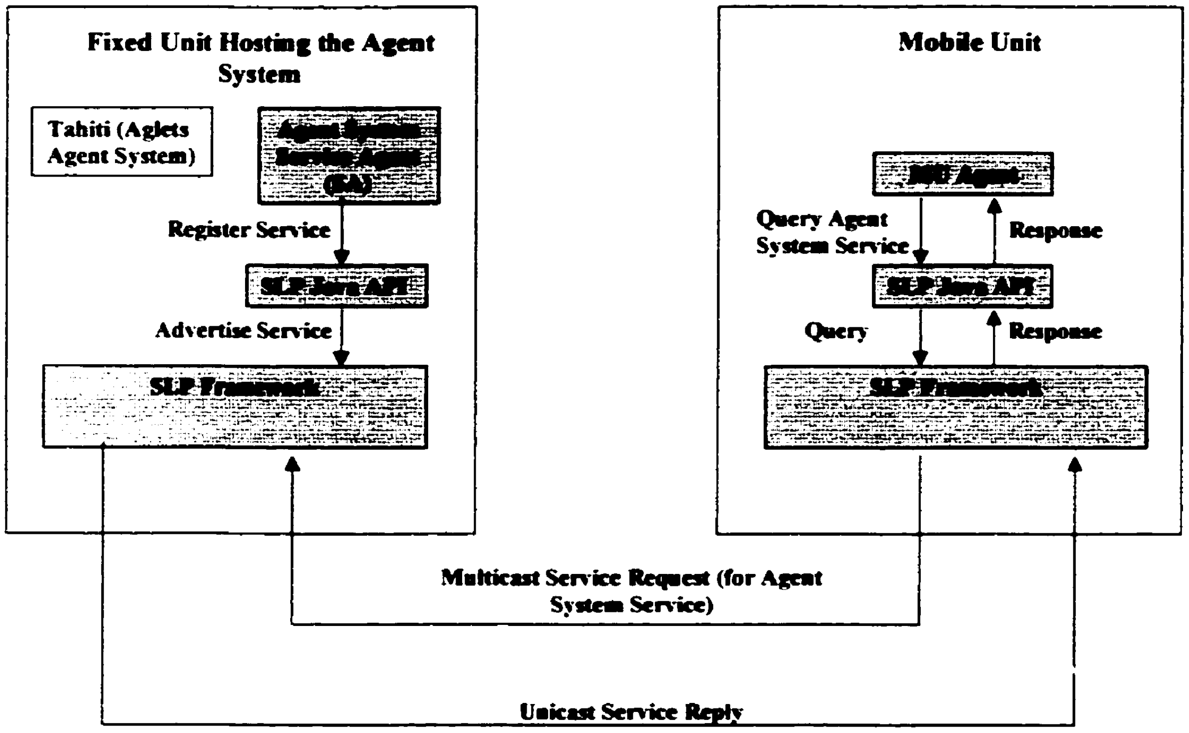

Fixed Unit Hosting the Agent System
Tahiti (Aglets Agent System)
Agent System Service Agent (SA)
Register Service
SLP Java API
Advertise Service
SLP Framework
Mobile Unit
MU Agent
Query Agent System Service
Response
SLP Java API
Query
Response
SLP Framework
Multicast Service Request (for Agent System Service)
Unicast Service Reply

Figure 4.2 - Interaction of SLP Components

4.3 Mobile Web Browser Support System

This section describes the mobile web browser support system developed for the prototype. The support system was written entirely in the Java programming language, chosen for its platform independence so that the FU agent would be portable to a wide variety of foreign networks. The benefits of Java as a language for mobile code have been well documented [Won99, Tho97]. In the following sections, the various components of the support system will be examined in detail.

4.3.1 MU Agent

4.3.1.1 Modules

The MU Agent consists of several modules (see Figure 4.3).

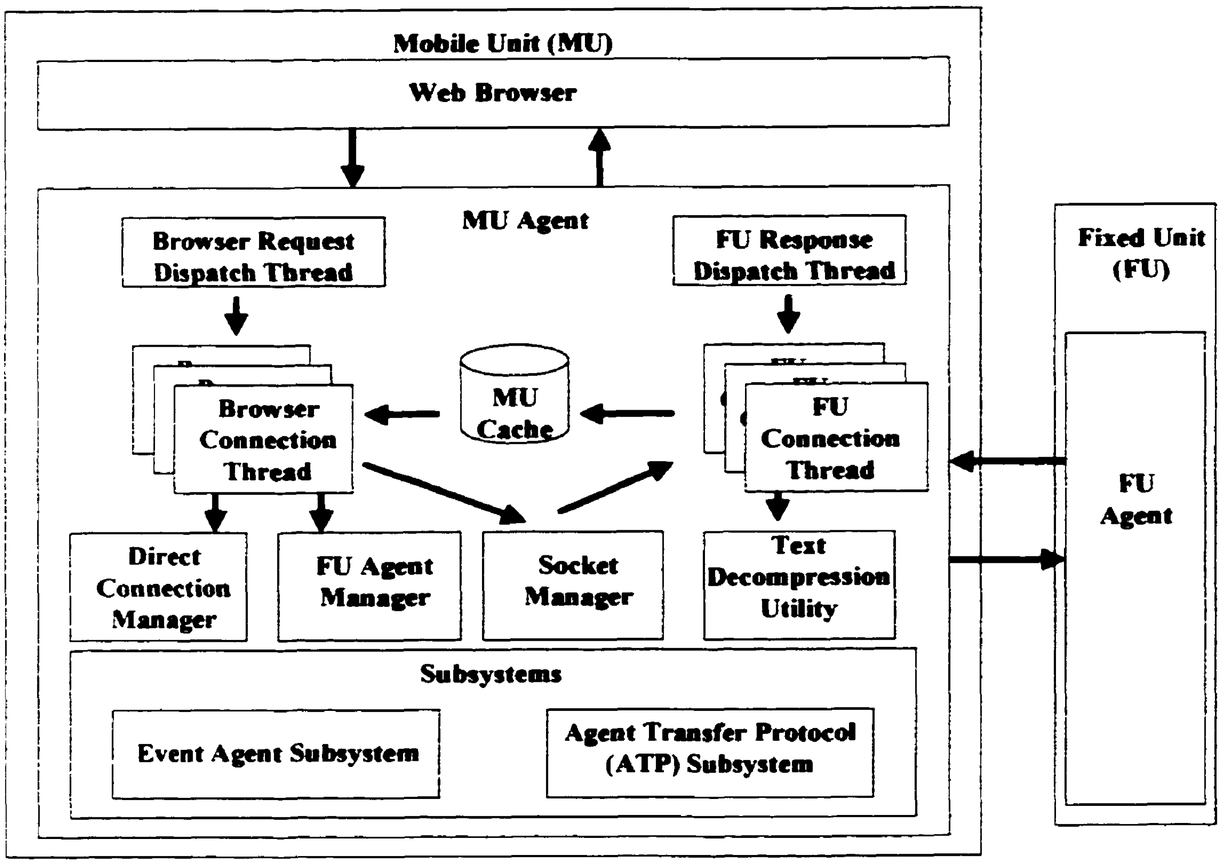

Mobile Unit (MU)
Web Browser
MU Agent
Browser Request Dispatch Thread
FU Response Dispatch Thread
Browser Connection Thread
MU Cache
FU Connection Thread
Direct Connection Manager
FU Agent Manager
Socket Manager
Text Decompression Utility
Subsystems
Event Agent Subsystem
Agent Transfer Protocol (ATP) Subsystem
Fixed Unit (FU)
FU Agent

Figure 4.3 - MU Agent Block Diagram

Browser Request Dispatch Thread – The browser request dispatch thread is responsible for accepting requests from the web browser and starting browser connection threads to handle the requests.

Browser Connection Thread – The browser connection threads are responsible for processing the requests received from the web browser. Processing involves checking the MU cache to see if a cached copy of the requested entity exists and returning the cached entity to the browser if it does. If the entity is not in the cache, the browser connection thread forwards the request to a FU agent (if one exists), or uses a direct connection to handle the request. Note that the browser connection thread will attempt to start a new FU agent if one does not exist.

MU Cache – The MU cache stores cached copies of information retrieved by the FU Agent. Entities in the cache are kept consistent by storing a time to live value with each entity. Any time an entity is retrieved from the cache, its time to live value is checked to ensure that the entity has not expired. If an entity has expired, it is removed from the cache. There is no size limit placed on the cache, and currently no replacement policy has been implemented. However, the cache is not persistent and is cleared each time the MU Agent is restarted.

FU Response Dispatch Thread – The FU response dispatch thread is responsible for accepting responses from the FU agent[1] and initiating FU connection threads to handle the processing of the responses.

FU Connection Thread – The FU Connection threads are responsible for handling communications from the FU agent to the MU agent. They read responses to web page requests sent to the FU agent, write the responses to the MU cache and forward them to the web browser. The FU Connection threads also assist in the setup and teardown of the connection between the MU agent and the FU agent.

Direct Connection Manager – The direct connection manager module uses the HTTP protocol to establish direct connections to web servers to retrieve entities requested by the web browser. These direct connections are fallback measures that are used in cases where a network connection is available to the MU agent but a FU agent cannot be started (e.g. because the foreign network does not support the mobility framework).

Socket Manager – The socket manager is used to maintain open connections from the web browser to the MU agent. Every socket from the web browser to the MU agent containing a request that is forwarded to the FU agent is saved. When a response to the request is received from the FU agent, the socket can be retrieved from the socket manager and used to transmit the response to the browser. It is necessary to save the socket connections since web browsers do not support asynchronous responses. The web browser expects to receive a response on the same socket that was used to transmit the request. Hence, the socket must be saved.

[1] In Figure 4.3 interactions between internal MU agent components and external processes (the web browser and the FU agent) are represented by single arrows between the MU agent and the processes

FU Agent Manager – The FU agent manager is responsible for dispatching FU agents into a foreign network and monitoring the FU agent state.

Text Decompression Utility – The text decompression utility is responsible for decompressing text that has been compressed by the FU agent before transmission. Refer to Section 4.3.3's optimizations for more details.

4.3.1.2 MU Agent Subsystems

Several subsystems exist that operate on behalf of the MU Agent. These subsystems are implemented as daemon threads that terminate whenever the MU Agent terminates.

4.3.1.2.1 Event Agent Subsystem

The event agent subsystem is responsible for monitoring the communications link quality between the mobile unit and the fixed unit. It detects changes in link quality (including disconnections) and informs both the MU and FU agents so they may adjust their processing accordingly. The mobile web browser support system prototype contains two implementations of the event agent subsystem: the *event agent* and the *event agent simulator*.

The event agent is capable of monitoring the communications link quality in real time. It operates in conjunction with a program called WaveMANAGER (which accompanies the WaveLAN network card used for the wireless connection). The WaveMANAGER program is configured to perform a link test (i.e. test the quality of the communications link) with the fixed unit every 100 seconds, and the results of these tests are written to a log file. The test results contain information about the communications link including the current link quality (good, acceptable, poor or no connection) and more detailed information such as signal strength and signal to noise ratio.

At regular intervals, the Event Agent reads this log file and searches for the most recent link quality indicator (good, acceptable, poor or no connection). The event agent compares this indicator with the link quality it has stored (i.e. the previous link quality read last time the log file was checked). If the link quality has changed, the event agent sends a message to the FU agent (assuming a connection is available), providing it with the new link quality. The event agent also informs the MU agent about the change. After reading the log file, the event agent deletes it to prevent the file from growing indefinitely and to reduce the amount of log data that must be searched the next time the file is read.

The event agent implementation used here is clearly not suitable for use with foreign networks, since it requires manual configuration of the WaveMANAGER program and knowledge of the foreign network. However, the event agent is only specific to the optimizations used in this prototype. It is not a part of the architecture defined in Chapter 3 and hence does not affect the flexibility of the system. The event agent was included in this implementation to experiment with environment aware adaptation.

For testing purposes, a second complete implementation of the event agent subsystem was developed: the event agent simulator. The event agent simulator works in the same fashion as the event agent but reads the link quality from special scenario files instead of from the WaveMANAGER log file. These scenario files contain lines formatted as:

"link quality", duration

where the link quality is one of good, acceptable, poor or no connection and the duration specifies a period of time for which the link quality is valid. For example, consider the following excerpt from a file:

"good", 100

"poor", 10

"good", 90

This excerpt is interpreted in the following way. The link quality is good for 100 seconds, after which it changes to poor. It remains poor for 10 seconds, and then becomes good again for another 90 seconds.

As with the event agent, the event agent simulator notifies both the FU Agent and MU agent whenever the link quality changes. The simulator allows complete user control over the link quality, which is useful when testing certain optimizations that were employed.

4.3.1.2.2 Agent Transfer Protocol (ATP) Subsystem

ATP [Lan97] is an application layer protocol developed for transferring mobile agents between agent systems. The protocol is used by IBM's Aglets agent system to retrieve code for a mobile agent from a remote location. It specifies two different message types: request and response. Request messages are used to dispatch agents, retract agents, send messages to agents, and fetch the classes required to execute an agent. Response messages are sent after processing a request message and contain the requested information and a status code [Lan97].

The ATP subsystem is run on the mobile unit as a part of the MU agent to handle requests for the FU agent class files (the source code required to execute the FU agent). The subsystem, implemented as the ATP daemon, handles only the minimal subset of the ATP protocol necessary for transferring the FU agent class files to an agent system running on the foreign network. Specifically, the daemon is only capable of servicing FETCH request methods [Lan97] (which request class files) for the FUAgent.jar file that contains the class files used for the FU Agent. Any FETCH methods for other class files will result in a response message with an error code of 301 (FORBIDDEN). All other types of request methods (DISPATCH, RETRACT and MESSAGE) are handled by returning a response with a 401 (NOT IMPLEMENTED) status code. In this manner, the ATP daemon is able to respond to any ATP requests that it receives with a valid ATP response message.

The ATP daemon operates in the following way. It listens on port 434 (reserved for mobile agents [Lan97] and used by ATP) for messages encoded using the ATP protocol. Once it receives an ATP request message, it checks the request type. If the type is not FETCH, a response message with a 401 status code is returned. If the type of request is FETCH, the file path indicating the requested file is examined. Any files other than FUAgent.jar result in a response message with a 301 status code. The reason for this is security. The MU Agent does not want to open up the filesystem of the mobile unit to outsiders through the ATP daemon. Hence, only the class files that implement the FU agent are accessible via an ATP FETCH request. If the requested file is FUAgent.jar (as will most often be the case, since it is unlikely that the mobile unit will receive any unsolicited ATP request messages), the file is returned in a valid response message with a 100 (OKAY) status code.

IBM's Aglets toolkit actually provides objects that implement the functionality of the ATP daemon. However, these objects did not work properly in Aglets version 1.0.3, which was used in this implementation. A future implementation might want to look at replacing the ATP daemon with some of the Aglet Server classes that implement the ATP functionality. One benefit of using a proprietary daemon is that finer grained control is provided over security (as seen above, where control over the FETCH method is exerted at the file level).

4.3.1.3 User Interface

User interaction with the MU agent is handled through the web browser via special pages that can be loaded in response to predetermined URLs. Currently the only control the user has over the MU agent is the settings of the user profile. A user profile is resident on each mobile unit and is made available to the MU agent and the FU agent. Based on the user profile settings and the communications link quality certain optimizations are performed by the FU agent to reduce the number of bytes transmitted across the wireless portion of the network. The user profile page is accessible to users by entering the URL http://profiles in the web browser. This special URL is trapped by the MU agent, which returns the profiles web page (See Figure 4.4). The profiles web page contains an HTML form interface that can be used to manipulate the user profile settings. Similar to the request for http://profiles, the MU agent traps the HTTP POST method invoked by submitting the form. The MU agent then extracts the profile settings, saves them and transmits them to the FU Agent. The functionality of the user profile is explored further in Section 4.3.3.2.

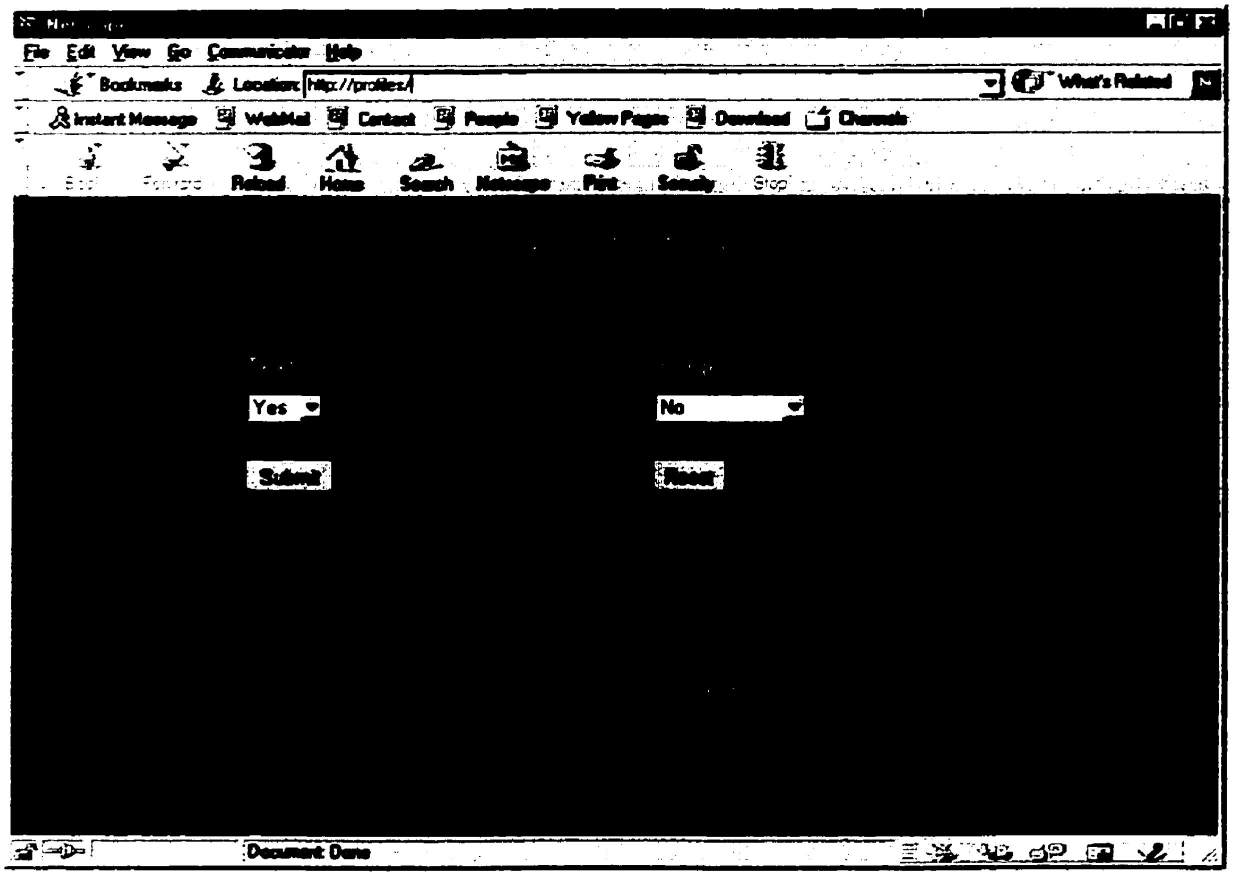


Figure 4.4 - User Profile Page Screen Shot

4.3.1.4 Operation

There are three major operations that the MU Agent is capable of handling. Together, these operations comprise the functionality provided by the MU agent.

4.3.1.4.1 Handling a Request from the Web Browser

The browser request dispatch thread listens on port 7000[2] (see communication, Section 4.3.4) for HTTP requests from the web browser. Whenever a request is received, a browser connection thread is created to handle the request. The browser connection thread reads the request header information to determine the URL of the requested entity. Processing now depends on the value of this URL.

[2] Port numbers are arbitrary.

The thread first checks if the URL corresponds to one of the predefined user interface URLs. That is, the thread checks if the URL is a request for the page http://profiles or a POST message from the profile web page. In the case of a request for the user profile web page, the current profile settings are read in from a disk file and used to update the profile page. The page is then transmitted to the web browser and the thread halts execution. For a profile POST message, the new profile settings are extracted from the message and written to a disk file (profiles are persistent over different sessions of the MU agent). Following this the profile settings are transmitted to the FU agent, if one exists. Note that the arrival of new profile settings will never initiate the dispatch of a FU agent (since a change in profile settings does not require any processing by the FU agent). As with a profile page request, the browser connection thread terminates after handling a profile POST message.

If the requested URL from the web browser does not correspond to a user interface URL, the MU cache is searched to see if it contains the requested entity. On a cache hit (i.e. the requested entity is in the cache and has not expired) the entity is read from the cache, transmitted to the web browser, and the thread terminates.

Following a cache miss, the browser connection thread must attempt to retrieve the requested entity from a web server. This requires a connection to the foreign network. The MU agent checks whether or not the mobile unit has an active network connection (by examining a parameter set by the Event Agent). For the purposes of this prototype, link quality values were generally simulated by the Event Agent Simulator component (although some tests used the WaveLAN link monitor utility to measure the link quality dynamically). If there is no network connection active, a message box indicating the error is presented to the user and the thread terminates. If a network connection is available, the thread continues to process the request.

In the case that the requested entity is an image (determined by the file extension), the thread checks to see if the referring page (i.e. the web page that contained the image) has been requested from the FU agent. If it has, then the image should be forthcoming and the thread terminates. The reasons for this are due to the single request for a web page optimization (see Section 4.3.3.4 for further details). If the referring page has not been requested, then the image will never arrive from the FU agent. The FU agent prefetches images based on the image URLs contained in the pages requested by the MU agent. The MU agent never forwards individual requests for images to the FU agent. Hence, if the page which refers to an image has not been requested from the FU agent, then the FU agent has no way of obtaining the URL for the image and will never retrieve the image. In this case, a direct HTTP connection is used to retrieve the image (bypassing the FU agent completely). Once the image is retrieved, the thread terminates.

In the case that the requested entity is not an image, the thread checks whether or not a FU agent exists on the foreign network. This check is performed by the FU Agent Manager object and is described in Section 4.3.1.4.3. Essentially the FU Agent Manager abstracts the details of dispatching and managing the FU agent on foreign networks. All the MU agent needs to be concerned with is that the FU Agent Manager will indicate if there is an active FU agent on the foreign network or not. Note that it is possible that the FU Agent Manager will put the browser connection thread to sleep while it attempts to dispatch a new FU agent into the foreign network.

If there is no FU agent running on the foreign network (and one could not be started), then a direct HTTP connection to the web server is used to satisfy the request from the web browser (since it is known that the mobile unit has an active network connection). A direct connection operates in the same way as a normal HTTP connection from the browser to the server with the added indirection that requests are first routed through the MU agent. No optimizations are performed. The direct connection retrieves the requested entity, transmits it to the web browser, and the thread halts.

If a FU agent is running on the foreign network, the HTTP request is reformatted using the proprietary protocol (see Section 4.3.3.5) and transmitted to the FU agent. The socket manager stores the socket from the web browser to the MU agent so that it may be used later to transmit the response from the FU agent back to the browser. All further processing associated with the request is handled by the FU Agent, so the browser connection thread terminates.

If at any point an error occurs while communicating with the FU agent, the error is logged with the FU agent manager.

4.3.1.4.2 Handling a Response from the FU Agent

The FU response dispatch thread listens on port 9000 for messages from the FU agent. When a message is received, a FU connection thread is started to process the response. There are five different types of messages that the FU agent may transmit. The processing associated with each of these messages is described here.

Case 1: If the FU connection thread receives a handshake message from a FU agent, this indicates that the FU agent has started execution. The thread records the port number on which the FU agent is listening and the identification number of the FU agent. Both values are contained in the handshake message (Section 4.3.2.2). The thread proceeds to transmit the current user profile and link quality to the FU agent (which completes the handshake process). Following this the thread notifies the FU Agent Manager that a FU agent has started executing. The manager will also wake up any sleeping browser connection threads.

Case 2: If the FU connection thread receives a message from the FU agent indicating that the FU agent is shutting down[3], the thread notifies the FU Agent Manager that the FU agent is no longer active.

Case 3: If the FU connection thread receives an HTTP entity (usually a web page, although other media types such as pdf files or audio files are possible), the thread checks the content type of the entity. If the content type is text/html, the thread invokes the text decompression utility on the entity to decompress the text and retrieve the actual web page. The thread proceeds to add the entity (uncompressed version for text) to the MU Cache (unless the entity is a response to an HTTP POST request which are not cached). Then the thread queries the Socket Manager to retrieve the web browser connection (socket) that was used to initiate the request for this entity. If an active socket is found (i.e. the web browser is still waiting for the entity) the thread sends the entity to the browser. If no socket is found or the socket is no longer active, no further processing is performed.

[3] See Section 4.3.2.2 for reasons why this would occur

Case 4: If the FU connection thread receives a set of images associated with a requested web page, it writes each image to the cache. The thread then displays a message box to the user, indicating for which page all the images have arrived. The message box is used to notify the user that a page has arrived and is available in the cache in case the user has moved to a different page while waiting for the requested page to load.

Case 5: If the FU connection thread receives an error message from the FU agent indicating that some error occurred on the FU agent while processing a request, the thread logs the error and checks to see if the socket corresponding to the requested web page for which the error occurred is available in the Socket Manager. If the socket is available and active the error message is transmitted to the web browser.

4.3.1.4.3 FU Agent Dispatch

Dispatch of a FU agent to an agent system on a foreign network always occurs in response to a request from a web browser. For example, a change in communications link quality or to a user profile will not initiate a FU agent dispatch even though changes in these values are transmitted to the FU agent if it already exists. The FU agent is never dispatched until it is absolutely required. Typically this occurs on the first webpage request, which adds some overhead but is necessary to avoid incurring the overhead in cases where the system is not used. The alternative approach would have been to always dispatch the FU agent when a foreign network is encountered. Such an approach wastes bandwidth if the user performs no web browsing on the foreign network.

The FU Agent Manager object handles FU agent dispatch. It is responsible for keeping track of the current state of the FU agent and managing any errors that occur while communicating with the FU agent. The FU agent manager associates four different states with the FU agent: *inactive*, *transmitting*, *active*, and *not allowed*. The *inactive* state means that there is no FU agent active in the foreign network. The *transmitting* state indicates that the FU agent manager is currently attempting to dispatch a FU agent to the foreign network. An *active* state corresponds to an active FU agent (i.e. the FU Agent Manager has successfully dispatched a FU agent to the foreign network). The *not allowed* state means that a FU agent cannot be dispatched on the foreign network because either: (a) an agent system capable of executing the FU agent cannot be found on the foreign network or (b) the number of times the FU agent has been restarted on a particular foreign network has crossed a predefined threshold value. Figure 4.5 shows the state transition diagram for the FU agent manager.

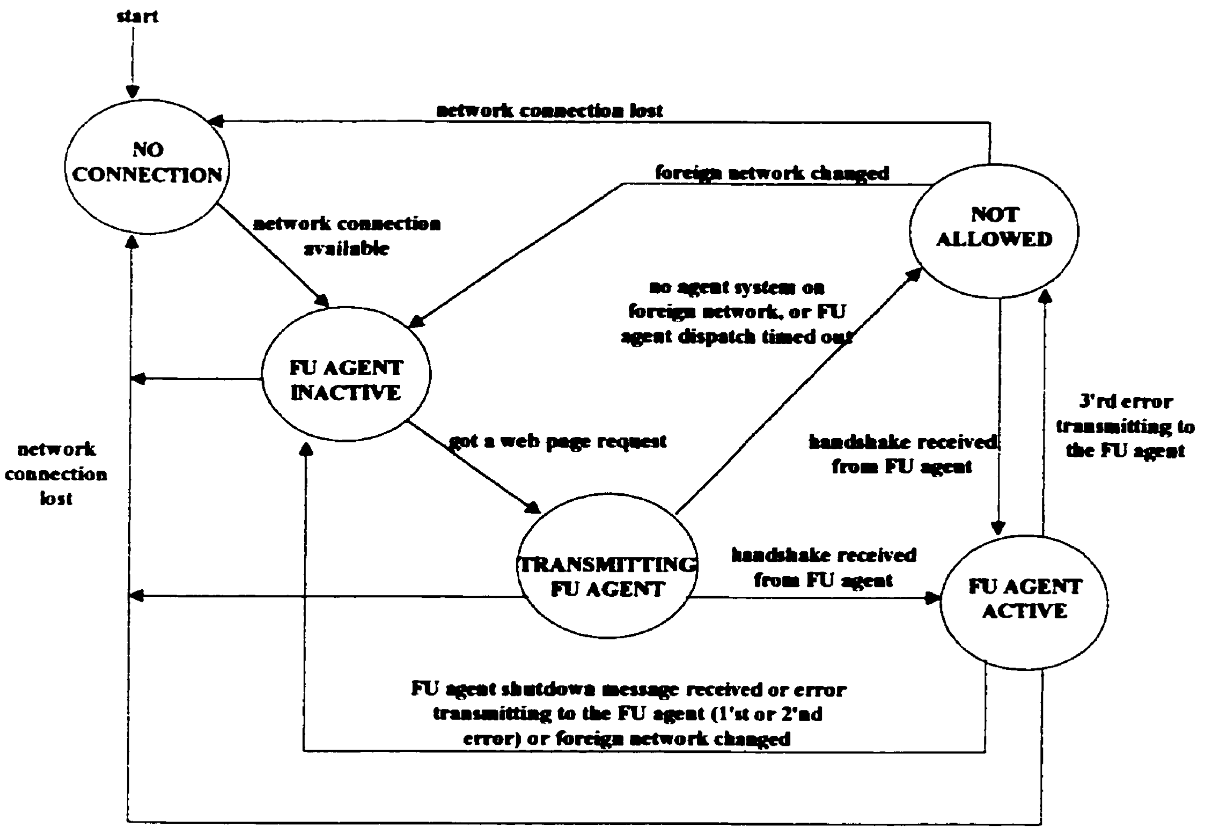

start
NO CONNECTION
network connection lost
foreign network changed
NOT ALLOWED
network connection available
no agent system on foreign network, or FU agent dispatch timed out
FU AGENT INACTIVE
network connection lost
got a web page request
handshake received from FU agent
3'rd error transmitting to the FU agent
TRANSMITTING FU AGENT
handshake received from FU agent
FU AGENT ACTIVE
FU agent shutdown message received or error transmitting to the FU agent (1'st or 2'nd error) or foreign network changed

Figure 4.5 - FU Agent Manager State Transition Diagram

Initially the FU agent manager is in a no connection state. This state indicates that the mobile unit does not have a network connection. As soon as the mobile unit is connected to a network (determined by the Event Agent) the FU agent manager associates an inactive state with the FU agent. The first time a browser connection thread receives a request from the web browser, it queries the FU agent manager to see if there is an active FU agent on the foreign network. When the query is received, the FU agent manager immediately moves to a transmitting state and blocks the browser connection thread that issued the query. Any further browser connection threads that query the state of the FU agent manager while it is in a transmitting state will also be blocked.

Once in the transmitting state, the FU agent manger tries to locate a compatible agent system on the foreign network using SLP via the SLP API (see Section 4.2.3 for more details). If an agent system is found, the manager dispatches a FU agent (with the mobile unit host address and a unique FU agent id as parameters) to the agent system and then waits for a handshake from the FU agent. The handshake is a notification message that will be received by a FU connection thread. When the handshake is received, the FU connection thread informs the FU agent manager and the manager moves into an active state. At this point, it wakes up all the sleeping browser connection threads and informs them that there is a FU agent active on the foreign network. Subsequent queries to the FU agent manager will immediately indicate that a FU agent is active as long as the FU agent manager remains in an active state.

If no agent system is found on the foreign network, the FU agent manager moves to a not allowed state (i.e. the current foreign network does not allow FU agents). All sleeping browser connection threads are woken up and informed that there is no FU agent, so they will revert to the default direct connection to satisfy the web browser requests. Subsequent queries to the FU agent manager will resolve to a not allowed response as long as the FU agent manager remains in the not allowed state.

It is important to note that while the FU agent manager is in a transmitting state the browser connection threads are not put to sleep indefinitely. Rather they have a timeout associated with them. If the timeout expires before the manager wakes them up, they will wake themselves up and cause the manager to move to a not allowed state (and processing will follow as indicated above). The reason for this is that the dispatch of the FU agent onto the foreign network may encounter problems that cannot be detected by the FU agent manager. The dispatch may seem to function correctly, but a handshake may never be received. In such a case, the browser connection threads should not wait indefinitely. The timeout period was chosen to give the FU agent manager a reasonable amount of time to dispatch the FU agent and receive a handshake notification. If the timeout expires, it is assumed that for some reason the FU agent is not allowed to execute on the foreign network so the FU agent manager is placed in a not allowed state. In the case where the timeout expires too early (i.e. the FU agent dispatch worked correctly, but took an unreasonably long time), the FU agent manager will move prematurely into a not allowed state. Fortunately, since the dispatch was error free, a handshake will be received from the FU agent and this handshake will move the manager from the not allowed state into an active state (and browser connection threads will now be informed that there is a FU agent active). Thus, as long as the dispatch is error free, there will be no problem. Eventually requests will go through the FU agent. One other benefit of the timeout is that it prevents the latency for a request from being unnecessarily high. Once the first timeout expires, direct connections will be used to satisfy requests until the FU agent is in place and transmits a handshake. Therefore, the waiting time for requests when a FU agent is dispatched is bounded by the timeout value rather than the actual dispatch time for the FU agent.

Once the FU agent manager is in an active state there are three events that can cause it to change its state again. First, if the foreign network with which the mobile unit is communicating changes (i.e. the mobile unit moves to a new location), the FU agent manager will move into an inactive state thereby allowing the FU agent to be dispatched into the new foreign network. Secondly, if the FU agent manager receives a timeout notification indicating that the FU agent has timed out and halted execution, the manager will also move to an inactive state so another FU agent may be dispatched. Finally, if an error occurs while the MU agent is communicating with the FU agent, the error will be logged with the FU agent manager and cause the manager to change states. This state change depends on how many errors have occurred. The FU agent manager maintains an error count and an error threshold that indicates the maximum number of errors that are to be tolerated on a particular foreign network. Each time an error is logged with the manager the error count is incremented. If the count is below the threshold value, the manager moves into an inactive state. If the count is greater than or equal to the threshold, the manager will move into a not allowed state. Effectively this limits the number of retransmissions of the FU agent that are allowed for a foreign network. Errors can be caused by various external factors beyond the control of the system. For example, an administrator on the foreign network may decide to terminate the FU agent for several reasons. If the FU agent is halted prematurely it may not have a chance to indicate to the MU agent that it is being terminated. The MU agent will detect the error when it tries to communicate with the FU agent and log the error accordingly with the FU agent manager. Since there is overhead associated with transmitting the FU agent, the number of transmissions should be limited on a particular foreign network. The error threshold scheme is used to provide this limit. For this prototype implementation the threshold value of 3 was chosen arbitrarily.

4.3.2 FU Agent

4.3.2.1 Modules

The FU agent is an aglet. It is composed of the modules shown in Figure 4.6 and described below.

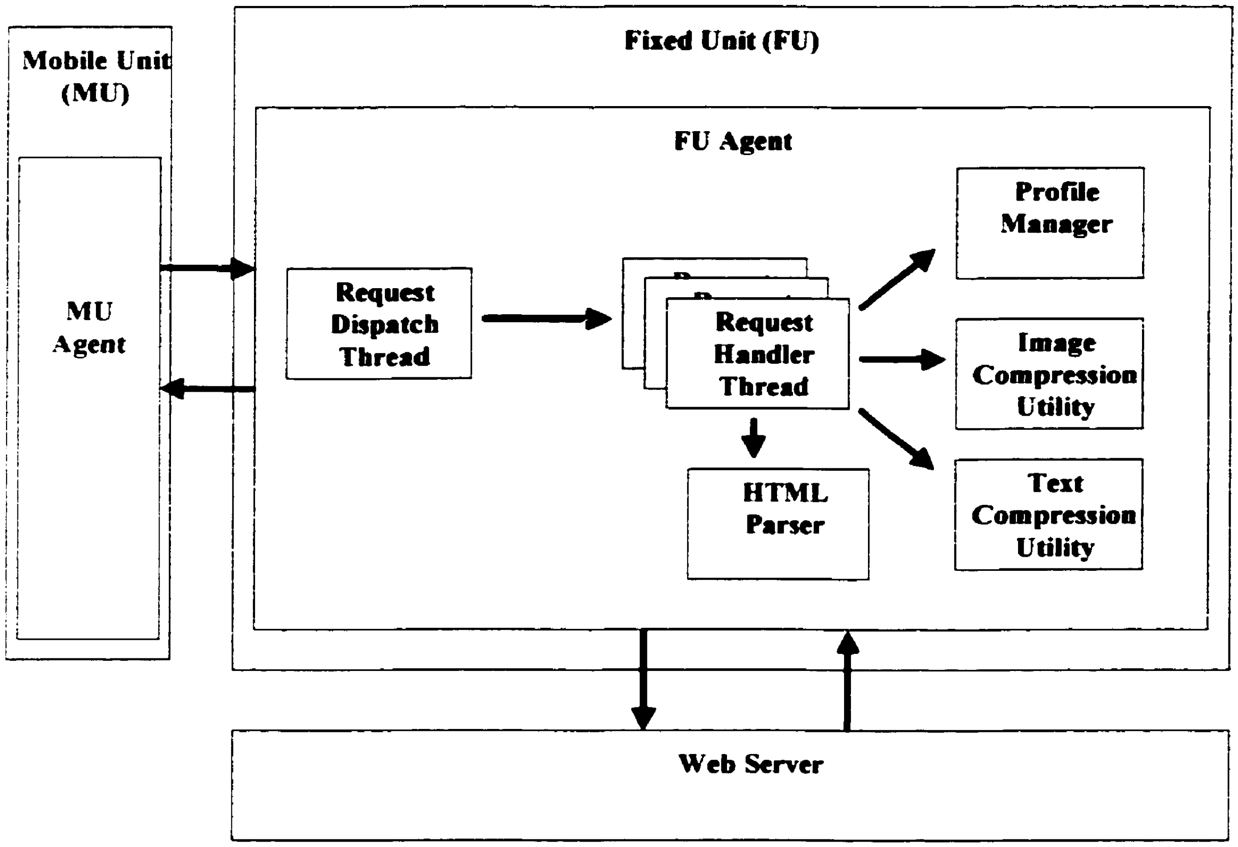


Figure 4.6 - FU Agent Block Diagram

Request Dispatch Thread – The request dispatch thread listens on a randomly selected port (the next section describes why a random port is chosen) for request messages transmitted by the MU agent. When a request is received, the dispatch thread creates a handler thread to take care of the processing associated with the request.

Request Handler Thread – The handler threads perform the actual processing associated with requests from the MU agent. They are responsible for contacting web servers to retrieve requested entities as well as transmitting those entities back to the MU agent. Handler threads also manage requests to change the user profile and communications link quality settings.

Profile Manager – The profile manager maintains the current user profile (see Section 4.3.3.2) and the communications link quality. It exposes the profile settings and link quality to all the request handler threads so that they may adjust their processing accordingly.

Image Compression Utility – The FU agent uses the image compression utility module to perform image optimizations. In this prototype, the image compression utility is only capable of resizing (compressing) image files encoded using the gif or jpeg file formats [Mur94]. The compression utility is used to make thumbnail sized versions of images before they are transmitted.

Text Compression Utility – The text compression utility is used to compress text entities (specifically web pages) before they are transmitted to the MU agent. Section 4.3.3 describes these optimizations.

4.3.2.2 Operation

Recall Section 4.3.1.4 describes how the MU agent locates an agent system on a foreign network and dispatches the FU agent to the agent system. Upon arriving at the agent system, the FU agent begins execution by parsing the initialization parameters that it is passed. These parameters include a unique identifier for the FU agent and the IP address of the MU agent (so that the FU agent knows the location of the mobile unit hosting the MU agent with which it is communicating). The FU agent then selects a port number at random on which it will listen for requests from the MU agent. The port number is not specified in advance because many FU agents could potentially be executing in a single agent system at the same time. Since many operating systems do not permit more than one process to listen on a single port, the port number is chosen at random to minimize conflicts.

After the port number is selected, the FU agent sends a handshake message to the MU agent. This message contains the identifier of the FU agent and the port number on which the FU agent will be listening. At this point, the FU agent is ready to handle requests from the MU agent so it starts the request dispatch thread, which listens to the chosen port for requests from the MU agent. A timeout value is also initialized for the FU agent. If the timeout period elapses without any requests from the MU agent, the FU agent transmits a shutdown notification to the MU agent and terminates. Each time a request is received from the MU agent, the timeout period resets itself. The timeout was included so that FU agents cannot execute indefinitely on the foreign network. Since disconnections are a frequent occurrence with mobile units, the FU agent cannot necessarily rely on the MU agent for a notification of when to quit. Hence a timeout is used to ensure the proper shutdown of all FU agents.

The request dispatch thread accepts requests transmitted by the MU agent. When a request is received a new request handler thread is started. Processing depends on the type of request and on the current settings of the user profile and communications link quality. In general, there are four types of requests that must be handled and they are discussed in the following sections.

4.3.2.2.1 Update User Profile

The MU agent transmits an update user profile request whenever the user changes their profile settings (Section 4.3.1.4.1). The handler thread reads the new profile settings from the request and updates the local profile settings stored by the profile manager so that the new profile settings are accessible to all handler threads.

4.3.2.2.2 Update Link Quality

The Event Agent subsystem transmits an update link quality request whenever it detects a change in the quality of the communications link between the mobile unit and the fixed unit. The handler thread reads the new communications link quality settings from the request and updates the local link quality settings stored by the profile manager.

4.3.2.2.3 Handle an HTTP GET Request

HTTP GET requests are messages transmitted by the MU agent that request a particular HTTP entity (e.g. a web page, a Java class file, etc). They have been reformatted using the proprietary protocol to include only the necessary information that the FU agent needs to retrieve the entity. The handler thread reads the request and contacts the web server (using HTTP) to fetch the requested entity. If an error occurs at any point while fetching the entity from the web server, the handler thread transmits an error message to the MU agent and terminates execution. Once the entity has been successfully retrieved the handler thread examines the entity content type. Any entity that has a content type other than text/html is simply transmitted back to the MU agent unmodified. If the entity is of type text/html (i.e. a web page) processing is more complicated and depends on the settings of the profile manager.

Case 1: Profile Manager indicates the user is interested in text only

Since the web page has been retrieved the handler thread has all information of interest to the user. The web page is compressed using the text compression utility and transmitted to the MU agent. No additional processing is required and the thread terminates.

Case 2: Profile Manager indicates the user is interested in text and images

Initially the thread compresses the web page and transmits it to the MU agent. This gives the user on the mobile unit some immediate feedback because they see the web page appear right away. Then the thread parses the web page to retrieve all the image tags contained in the HTML (these are tags that look like <img src = ...>). A URL is constructed for each of the images referenced by the web page. This involves converting the relative URLs contained in the web page to absolute URLs. Relative URLs provide only some of the URL information (like a relative file path), assuming that the unspecified parts may be obtained from the URL of the web page in which they are referenced. Before the thread may retrieve the images from the web server these relative URLs must be converted to absolute URLs, so that the handler thread knows the precise location of each of the images.

Example: Suppose a web page with the URL http://www.cs.umanitoba.ca/index.html contains an image tag <img src = title.gif>. The handler thread will use this information to construct the absolute URL http://www.cs.umanitoba.ca/title.gif to load the image from a web server.

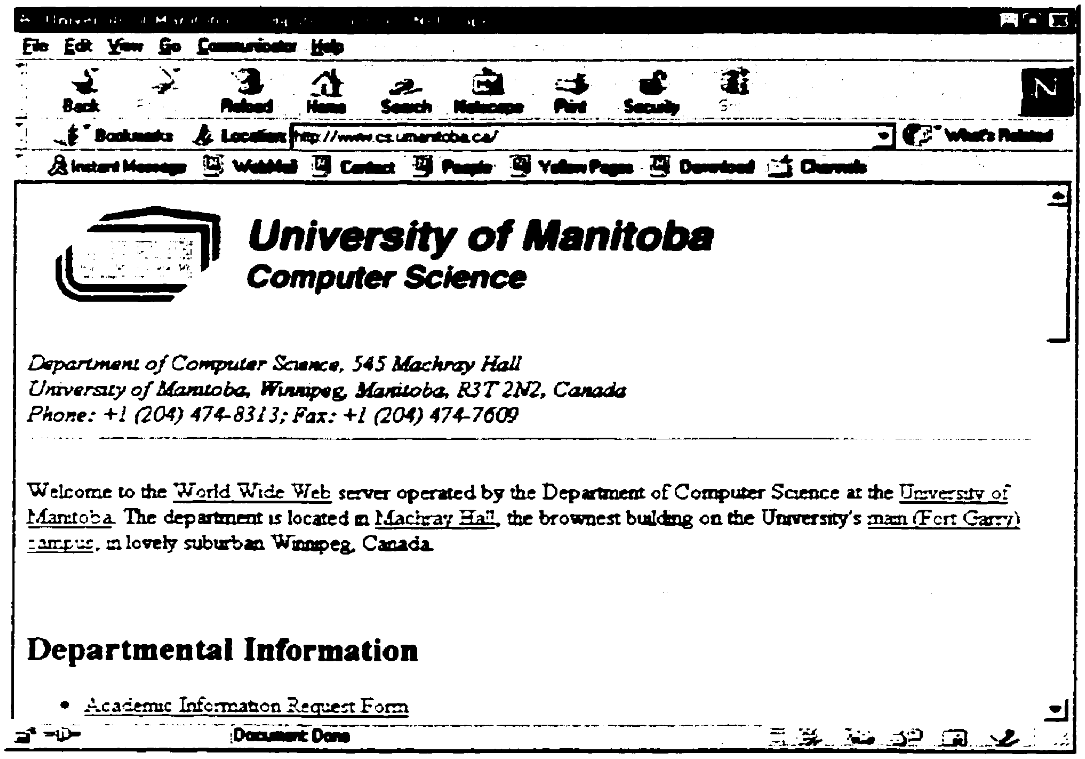
Once all of the image URLs have been extracted from the web page the images are retrieved from web servers. Processing again differs here based on the current settings in the profile manager. If the profile manager indicates that the link quality is good and/or that no image optimizations are to take place, then the images are transmitted together to the MU agent and the thread terminates. However, if the profile manager indicates that the link quality is poor and that images may be compressed, then the image compression utility is invoked on each of the images. The image compression utility is only effective on images encoded using the gif or jpeg image file formats. It takes an image and reduces it in size (makes a thumbnail of the image) to decrease the number of bytes used to encode the image. Once the images have been optimized they are transmitted to the MU agent and the handler thread terminates.

Case 3: Profile Manager indicates that the user is interested in images only

In this case, the thread does not transmit the web page to the MU agent. Instead it parses the web page to find all the image tags and all the anchor tags (links, tags that look like <a href = ...>) contained in the HTML. The thread constructs a new web page of the format:

```
IMAGES
Image 1
Image 2
...
LINKS
Link 1
Link 2
...
```

This new page contains all of the links and images from the requested page but none of the text (with the exception of the link descriptions). Figures 4.7 and 4.8 show an example of a complete web page and the same page after applying the links and images only optimization. The new page is compressed and transmitted to the MU agent. Processing now proceeds in exactly the same way as Case 2. Images are retrieved, optimized if necessary, and transmitted to the MU agent. After transmitting the images the handler thread terminates.



University of Manitoba
Computer Science

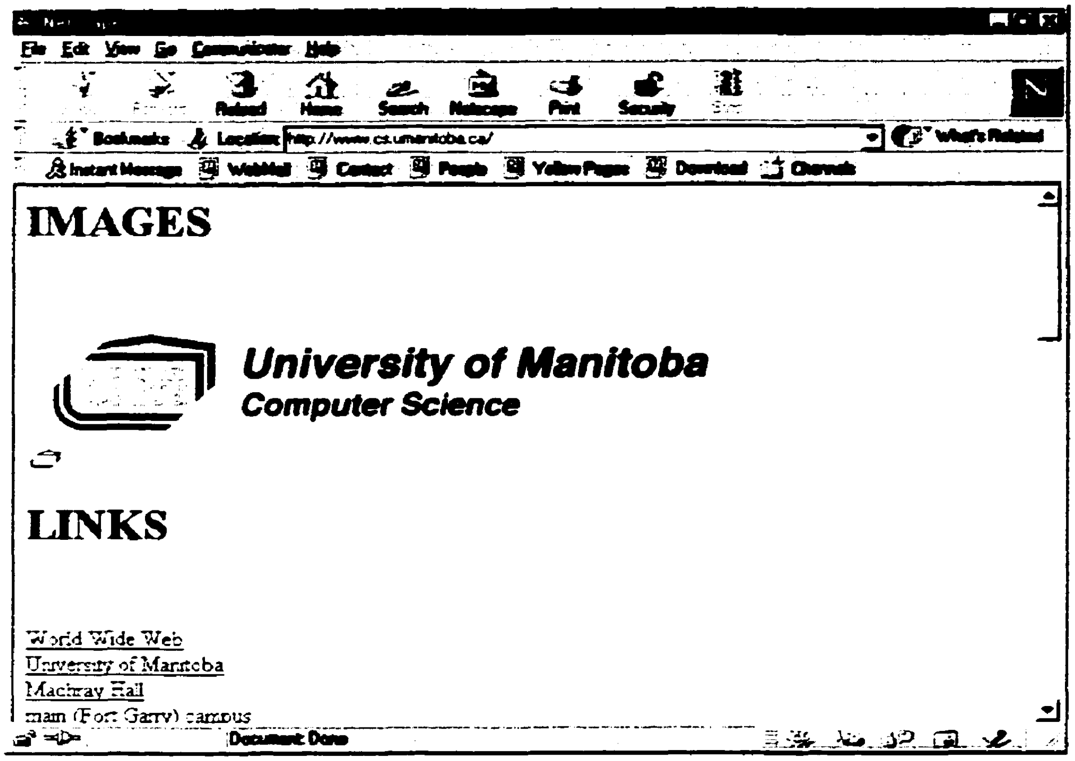
Department of Computer Science, 545 Machray Hall
University of Manitoba, Winnipeg, Manitoba, R3T 2N2, Canada
Phone: +1 (204) 474-8313; Fax: +1 (204) 474-7609

Welcome to the World Wide Web server operated by the Department of Computer Science at the University of Manitoba. The department is located in Machray Hall, the brownest building on the University's main (Fort Garry) campus, in lovely suburban Winnipeg, Canada.

Departmental Information

- Academic Information Request Form

Figure 4.7 - Sample of a Complete Web Page



IMAGES

LINKS

World Wide Web
University of Manitoba
Machray Hall
map (Fort Garry) campus

Figure 4.8 - Sample of a Web Page with Images and Links Only

4.3.2.2.4 Handle an HTTP POST Request

The MU agent transmits HTTP POST requests whenever a POST request is received from the web browser. They are handled in much the same way as HTTP GET requests except for one minor difference. POST requests are tunneled in the proprietary protocol because the data is too arbitrary for the protocol to effectively encode them. Therefore, rather than composing an HTTP request using the information in the MU agent request, the handler simply extracts the tunneled HTTP POST request from the MU agent request and transmits it directly to the web server. Responses are in the form of web pages and are handled in exactly the same fashion as responses to HTTP GET requests. That is, parse the page for images, compress and transmit the page, retrieve and transmit the images.

4.3.3 Optimizations

The prototype employs several different optimizations in an attempt to increase the efficiency of the communication at the application layer across the wireless portion of the network. This section highlights the various optimizations and discusses their operation. Note that these operations are specific only to this prototype, not the architecture.

4.3.3.1 Compression of Web Pages

The FU agent compresses all web pages before they are transmitted to the MU agent, which decompresses them upon reception. The compression technique employed in the prototype is Lempel-Ziv-Welch (LZW) compression. LZW compression was chosen for efficiency and simplicity. The algorithm (described in [Nel89]) is capable of compressing text in a single pass, without performing any advance analysis of the text. Further, decompression is performed on the compressed text without requiring any additional coding information (which means less overhead is required to transmit the compressed text). Compression is fairly efficient and the size of the code required to implement the compression is not prohibitive. Nevertheless it should be noted that this operation does place a requirement for a certain amount of processing power available on the mobile unit. It has been reported that using this technique *"compression levels of 50% or better should be expected"* [Nel89]. However, the actual compression achieved depends to a large extent on the redundancy of the text being transmitted.

4.3.3.2 User Profile

A user profile, which is resident on the mobile unit, allows users to select their preferred type of data and indicate their tolerance for certain optimizations that will decrease the quality of the information displayed by the browser. The profile gives users the ability to indicate what type of information they are interested in viewing (e.g. text only, images and links only, etc.). Whenever the profile is changed using the MU agent interface (Section 4.3.1.3), the MU agent transmits a new copy of the profile to the FU agent so it always possesses an up to date copy. Using the profile, the FU agent is able to determine exactly what type of information the user is interested in and can filter out any unimportant information. Thus, the FU agent is able to ensure that bandwidth is not wasted on information that the user is not interested in viewing. For example, if the user profile is set to text only, the FU agent will only transmit the text for web pages to the MU agent. It will not bother retrieving any of the images contained in the web pages because it knows (via the profile) that the user is not interested in them.

The user profile employed in this prototype is a very coarse granularity. It only allows users to filter out text or images (along with setting acceptable image quality levels). However, the profile could easily be extended to a much finer grain to allow filtering of many different media types including sound files, Java class files, etc. Currently the profile settings are grouped into three broad categories: text and images, text only, and images and links only. The first category, text and images, is the most common. Here all the text and images contained on a requested web page are returned to the user. The second category, text only, works in the same way as the text only option supported by most web browsers. It was included for consistency with current web browser systems. The third category, images and links only, returns only the images and links from a requested web page (text is omitted). It is useful in situations where the user is searching for images (e.g. searching for clipart) or where the user wants to retrieve an image from a known web page (i.e. the URL is known in advance) without retrieving additional (and unwanted) text from the page. Within each category, the user may specify whether or not image compression is permitted. Table 4.1 summarizes all the different profile settings available in the prototype and explains their effect on the processing performed by the FU agent.

Table 4.1 - User Profile Settings

Profile Settings		**Description**
Text	**Images**	
Yes	Yes	The FU agent retrieves the requested web page and all the images for that web page. Both the text and images are transmitted to the MU agent.
Yes	No	The FU agent retrieves only the requested web page and sends it to the MU agent. Images are not retrieved.
Yes	Optimized - Compressed	Same as for the case with Text = Yes and Images = Yes, but if the communications link quality is poor then the images are compressed to thumbnail size before they are transmitted
No	Yes	The FU agent retrieves the requested web page and all the images for that web page. The web page is not transmitted to the MU agent. Instead, the FU agent dynamically constructs and sends a new web page containing only the links and the images from the requested page. The FU agent then transmits the images.

Profile Settings		Description
Text	Images	
No	Optimized – Compressed	Same as for the case with Text = No and Images = Yes, but if the communications link quality is poor then the images are compressed to thumbnail size before they are transmitted

4.3.3.3 Environment Aware Adaptation

The prototype makes use of network statistics (specifically the quality of the communications link) to improve the efficiency of transmissions. The Event Agent subsystem (recall Section 4.3.1.2.1) runs as a part of the MU agent and continuously monitors the quality of the communications link between the mobile unit and the fixed unit. Whenever this link quality changes an update is sent to the FU agent. Based on this link quality and the settings of the user profile, the FU agent is able to apply tradeoffs in the fidelity of the image data to make better use of the available bandwidth. The tradeoffs used in this prototype operate on the dimension of image size. When link quality is poor and the user has indicated that image optimizations are allowed, images are reduced to thumbnail size before transmission. Such an optimization significantly reduces the number of bytes transmitted by the FU agent.

4.3.3.4 Single Web Page Request

Currently web browsers running the HTTP1.0 protocol open up a separate TCP/IP connection to a web server for each image referenced on a web page. This involves a good deal of unnecessary overhead. Note that this problem has largely been solved by the HTTP1.1 protocol [Fie97], but it is not yet widely implemented in commercial web browsers. To overcome this limitation of HTTP1.0 the prototype system supports a single request for a web page and all the images that it contains. The MU agent transmits a URL for a web page in a request message to the FU agent. Once the FU agent has this URL, it uses the user profile settings to determine what information (text, images, both) the user wants. The FU agent proceeds to fetch the desired information independently and transmits it back to the MU agent. For the case of images, this involves parsing the web page to find references to the images. On the MU agent side, subsequent requests made by the web browser for images contained on a page that has been requested from the FU agent are blocked. With the initial URL and the user profile, the FU agent has all the information needed to process the request and return the entities of interest to the user.

4.3.3.5 Proprietary Protocol

When transmitting information across the wireless portion of the communications link the MU and FU agents utilize a proprietary protocol. HTTP requests from the web browser are reformatted using this protocol before being transmitted to the FU agent. Similarly, the MU agent converts FU agent responses encoded using the proprietary protocol back into HTTP before forwarding them to the web browser. The protocol used attempts to be less verbose than the HTTP protocol and to reduce the header information that must be transmitted. Additionally, the protocol employed here treats all images referenced by a web page as a single entity and transmits them together. The proprietary protocol is described in detail in Appendix B.

4.3.4 Communication Protocols

This section summarizes the communication protocols used between the various components of the prototype.

4.3.4.1 Web Browser to MU Agent

Communication between the web browser and the MU agent is done using the HTTP1.0 protocol. The web browser is configured to use a local proxy on port 7000. This redirects all the HTTP requests transmitted by the browser to TCP port 7000 on the mobile unit. The MU agent listens on this port and intercepts HTTP requests that the browser transmits. Depending on the type of request, the MU agent either closes the TCP connection (socket) or saves it until a response is received from the FU agent so that the requested entity can be forwarded to the browser.

4.3.4.2 FU Agent to MU agent

The FU and MU agents communicate using the proprietary protocol (recall Section 4.3.3.5).

4.3.4.3 FU Agent to Web Server

The FU agent communicates with web servers using the standard HTTP1.0 protocol. Requests that the FU agent receives from the MU agent which are encoded using the proprietary protocol are reformatted as HTTP requests and transmitted to a web server.

4.3.4.4 MU Agent to Agent System Service Agents

The MU agent communicates with SLP service agents using the SLPv2 protocol as defined in RFC 2608 [Gut99b]. In these cases, the MU agent is playing the role of the user agent.

4.3.4.5 Agent System to ATP Daemon

Communication between the aglets agent system and the ATP Daemon module (in the MU agent) occurs using the ATP (agent transfer protocol) version 1.0 [Lan97].

4.4 Implementation issues

This section examines the Mobile IP triangle routing problem (Section 2.2.2) with respect to the architecture presented in Chapter 3. The impact of the problem on the architecture is discussed and insights are provided into potential solutions.

4.4.1 Mobility Support Using Mobile IP

Although Mobile IP may be used within the mobility framework to provide support for mobility of nodes, it is not particularly well suited for the architecture described here. The general architecture presented in this thesis tends to exacerbate the problem of triangle routing that occurs in current implementations of Mobile IP [Sol98] [Per98].

Recall from Section 2.2.2 the discussion of Mobile IP. Triangle routing arises from the inability of a mobile unit to inform a correspondent node (a node with which the mobile unit is communicating) the mobile unit's current "care of address" [Per98] [Sol98]. This prevents the correspondent node from using an optimized route when returning packets to the mobile unit. The situation is depicted in Figure 4.9. Packets from the mobile node to the correspondent are routed in the normal manner via the usual IP routing. However, packets from the correspondent node cannot be routed directly to the mobile unit, since the correspondent node does not know the mobile node's current point of attachment (current "care of address"). The IP packets that the correspondent node receives contain the mobile node's home address as the source address. Hence, all packets destined for the mobile unit will be routed to the mobile unit's home network (by way of the home address) and then tunneled to the current point of attachment (the care of address) via the home agent on the mobile unit's home network.

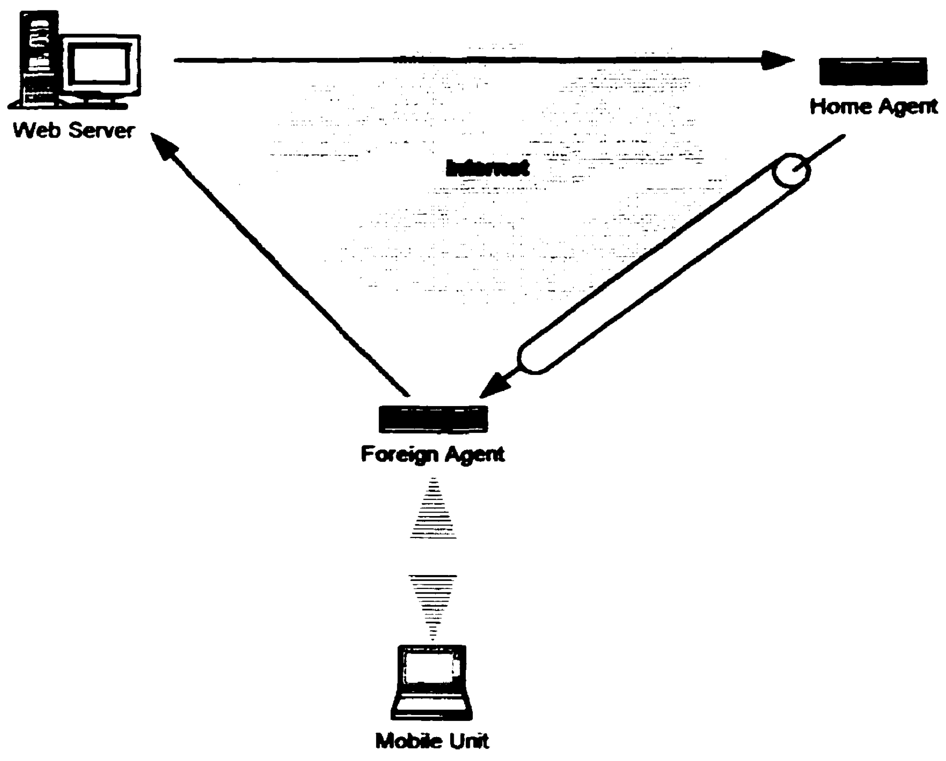


Figure 4.9 - Mobile IP Triangle Routing

Now consider the situation that occurs when the mobile web browser component of the prototype is used in conjunction with a foreign network that employs Mobile IP to provide the mobile unit with network connectivity (refer to Figure 4.10). Communications between the FU agent and the web server will occur in the normal fashion using the regular IP routing. This happens because the FU agent is executing on a fixed node within the foreign network, which is configured with a valid IP address for that subnet. However, observe the communications between the MU agent (on the mobile unit, which obtained a care of address from a foreign agent on the foreign network) and the FU agent. Packets sent from the MU agent to the FU agent will contain a source IP address equal to the mobile unit's home address (on the mobile unit's home network). Hence, whenever the FU agent attempts to send a response back to the MU agent, the response will be routed through the mobile unit's home network. This will occur even though the FU agent and the MU agent are executing within the same subnet, since the mobile unit on which the MU agent is operating continues to use its home address as the source address for IP packets (and, hence, packets will always be routed through the mobile unit's home network as described in the preceding paragraph). The one exception to this is the case where the FU agent is executing on the same host that is acting as a foreign agent since the foreign agent knows that packets destined for the mobile unit are to be sent out on the local subnet. Depending on the implementation of Mobile IP, the foreign agent may directly forward the packets to the mobile unit without going through the home agent[4]. This exception does not alleviate the problem because there is no guarantee that the agent system hosting the FU agent will reside on the same host as the foreign agent. In fact, this will often not be the case. Consider the situation where the foreign agent is not a host at all, but a router. Furthermore, the prototype described in this implementation functions at the application layer and has no provisions for addressing problems that arise at the network layer (such as triangle routing).

[4] This depends on whether or not the Mobile IP implementation modifies the routing table of the foreign agent to include an entry for the mobile unit. The mobile IP standard [Per96] does not require that this is the case although examples presented by Solomon [Sol98] suggest that it is possible.

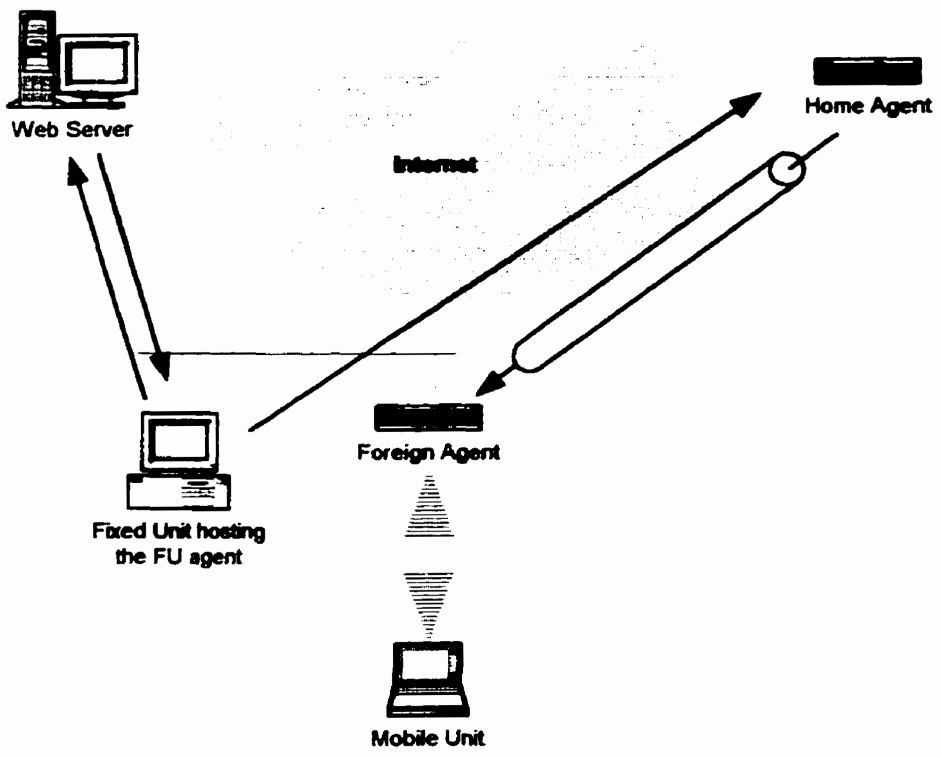


Figure 4.10 - Mobile IP with Dynamic Client/Intercept/Server Architecture

There are two potential solutions to this problem. The first is route optimization in Mobile IP [Per98] (from Section 2.2.2). The basic idea behind this is that the mobile unit could indicate its current "care of address" to the correspondent node. In this case, the correspondent would tunnel packets directly to the care of address without using the mobile node's home agent as an intermediary. The main problem with such a scheme is security. How to authenticate the binding update of the mobile node's current care of address with a correspondent node is an open question. Although there is work being done on this topic, it is not considered a priority because it is generally assumed that the mobile node will remain in relatively close proximity to its home agent and home network [Sol98]. Hence, the benefits of using an optimized route (which in many cases is not significantly shorter than the triangular route) are often not great enough to justify the cost of security.

A more promising solution is offered by the mobility support provided as a part of IPv6 [Per96b] (recall Section 2.2.3). Mobile IPv6 defines a data structure known as a binding cache that is implemented by each IPv6 node. The binding cache is used to store bindings where a binding is an association between a mobile node's home address and care of address, and the duration of time for which the association is valid [Per96b]. Whenever a correspondent node transmits a packet, it checks its binding cache for an entry corresponding to the destination address. If an entry is found the packet is tunneled to the care of address specified in the entry using an IPv6 routing header. Hence, the packet is automatically tunneled to the care of address rather than using a triangular route through the mobile node's home agent. Mobile IPv6 also defines a binding update option, which can be used to update the binding of a mobile node's care of address in the binding caches of correspondent nodes. Furthermore, IPv6 specifies an authentication header, which can be used to authenticate the binding update, thereby solving the security problems that occur when attempting to perform route optimization in Mobile IP [Per98].

The use of IPv6 effectively solves triangle routing at the network layer, isolating higher layers from the problem. This makes IPv6 a viable alternative to DHCP for inclusion in the mobility framework. Communications between the MU and FU agents could use the optimal route (see Figure 4.11) as they do with DHCP. The main drawback of IPv6 is that it has yet to be deployed in the larger Internet.

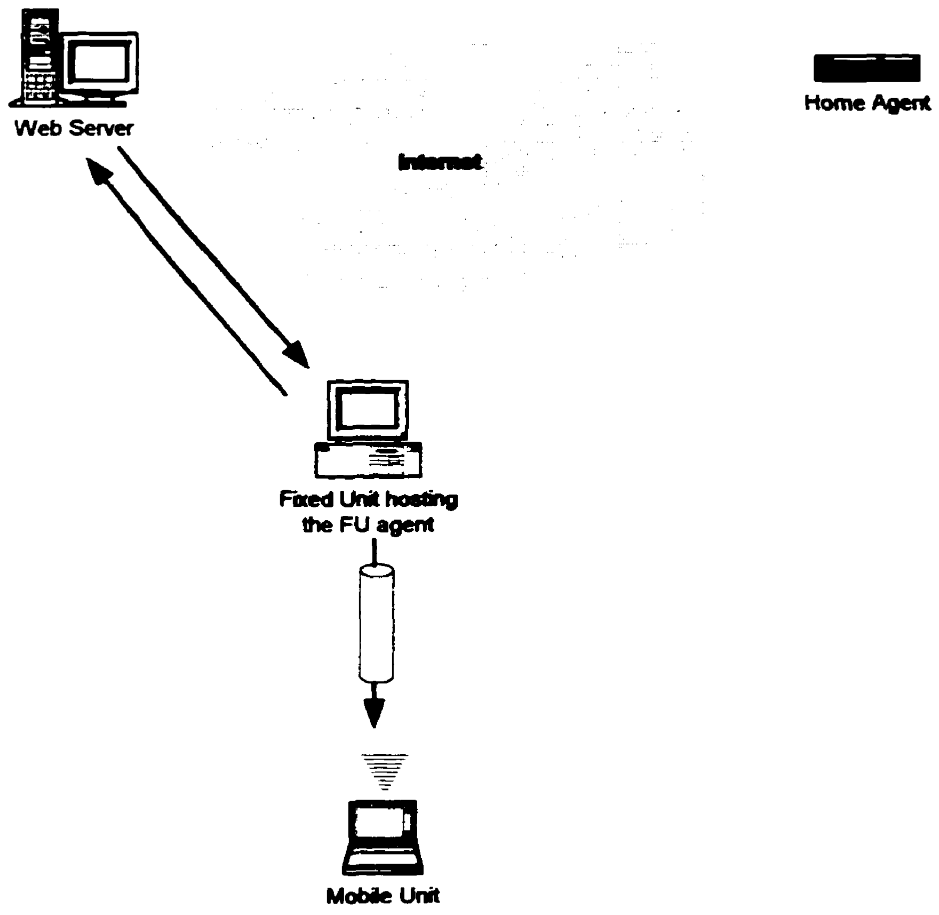


Figure 4.11 - Mobile IPv6 with Client/Intercept/Server Architecture

4.5 Comparison With Related Implementations

This section presents a comparison of the prototype implementation discussed previously with existing implementations of mobile web browsing systems. The related systems examined here were discussed in Section 2.4. In this section, the systems will be compared on the basis of the optimizations that they are capable of performing, their suitability for deployment in mobile computing environments, and their impact on the existing Internet architecture.

4.5.1 Proxy Filters

The proxy filter system proposed by Zenel [Zen99] employed an architecture similar to the one described here. However, the implementations are substantially different. The proxy filter system used the Columbia Mobile IP protocol (which is different from the standard Mobile IP protocol, described by Perkins [Per96]) to support mobility of nodes. Furthermore, the filter system required changes to this protocol to operate correctly. Specifically, the Columbia Mobile IP protocol was altered so that all data passes through a particular router, namely the mobile support router (MSR). This ensures that data destined for the mobile unit cannot avoid the filter (wired side proxy) by traversing an alternate route to the mobile unit that skips the node hosting the filter execution environment. The selection of the MSR is performed as a part of the modified Mobile IP protocol (dubbed PMICP for Proxy Mobile Internetworking Control Protocol). On the other hand, the prototype implementation described here works with existing, published standards. Either Mobile IP or DHCP may be used to support the mobility of the mobile units. Additionally, the discovery of the execution environment is decoupled from the mobility mechanism by using the SLP protocol and modeling the execution environment (here, an agent system) as a service offered by a foreign network.

The proxy filter system implementation supports native/binary execution environments (i.e. downloadable compiled code) [Zen99] rather than the interpreted execution environments (e.g. Java) used here in the prototype implementation. The filters employed in the proxy filter system are actually precompiled binary applications. This presents a distinct speed advantage over the interpreted format. However, there are three disadvantages in this approach. First, there is a major security problem [Zen99]. Binary applications cannot easily be verified to ensure they perform no malicious operations. They essentially have free reign over the system on which they are executing. Interpreted execution environments allow the downloaded code to be checked before it is executed. Furthermore, security policies (similar to those provided by the Aglets agent system) may be imposed on the downloaded code such as restricting access to the local filesystem. The second disadvantage with binary execution environments is the lack of flexibility [Zen99]. When using binary code, platform independence is sacrificed. The entire proxy filter system is not portable to different networks (without reworking the system for those networks). Also, filters designed to work for a particular architecture (e.g. on a Windows based network) will be unable to operate in other architectures. If the mobile unit wishes to interact with a number of networks varying in configuration, it would be necessary to equip the mobile unit with the required filters for each of those networks. The final disadvantage deals with the size of the filters (or agents). Typically compiled programs are much large than interpreted bytecode files (Java class files, in particular, may be further reduced in size if they are transmitted via a compressed jar, Java archive, file) [For94]. Although the prototype implementation loses the speed advantage over the proxy filter system, it gains the benefits of security, flexibility, and reduced filter (agent) size.

With respect to optimizations performed, an example created using the proxy filter system to optimize HTTP transmissions only implements a subset of the optimizations performed by the prototype. Specifically, it compresses the HTTP text data before transmission and uncompresses it using another proxy executing on the mobile unit [Zen99]. The prototype implementation performs this same optimization and others discussed in Section 4.3.3. However, such a comparison of optimizations is not really fair in the case of the proxy filter system since it was designed to support a much broader range of applications than web browsing and the example used for comparison was implemented to demonstrate how the system could be employed to increase the efficiency of HTTP transmissions. The proxy filter system could be extended to perform other optimizations similar to those implemented by the prototype.

4.5.2 Mowser

The Mowser system [Jos99] presents a much more diverse range of optimizations than the prototype implementation. In particular, it provides support for both proxy based and end-to-end optimizations. End-to-end optimizations are accomplished through content negotiation with the web server and allow the mobile unit to specify suitable data formats (e.g. request lower resolution pictures to make better use of the available bandwidth). These negotiations are carried out via modifications to the HTTP protocol to indicate what sort of content is acceptable for the mobile unit. This modification of the HTTP protocol implies that this particular optimization does not fit well within the existing

Internet architecture. However, as the authors point out, content negotiation is supported in HTTP 1.1, so the system could conceivably be made to function within the existing Internet architecture as support for HTTP1.1 becomes more widespread [Jos99].

The other optimizations performed by Mowser are proxy based. They include transcoding multimedia (which involves varying the fidelity of images and video, or converting the images or video into different coding schemes more suitable for wireless networks), elimination of active content (removal of Java Script, Java, etc. from web pages by the proxy if the mobile unit cannot support such content), elimination of HTML tags the mobile unit does not understand and personalization of data transmitted by the proxy (e.g. specifying different fidelity levels for images) [Jos99]. These optimizations are similar to those performed by the prototype system but are much more powerful. The concept of varying data fidelity is applied to a wider range of media types (e.g. video) and the user profile is extended to a much finer granularity. Furthermore, the proxy itself is able to filter out content that is undesirable (e.g. active content) or unusable (e.g. tags that are not supported). A similarity between the implementation of these optimizations is that both Mowser and the prototype rely on the ability of the proxy (or FU agent) to parse web pages to either filter content or retrieve desired content. Due to architectural differences (i.e. single proxy vs. two proxy), the Mowser system is unable to support compression of web pages and other lossless techniques to reduce bandwidth that require a corresponding agent executing on the opposite side of the wireless link without requiring changes to the web browsers themselves. Hence the optimizations performed by Mowser are typically lossy (such as dropping frames from a video). The one exception is the recoding of video into different formats that are more suitable for use on wireless networks [Jos99]. For this optimization, a plugin was developed and installed on the mobile unit to decode the video. This plugin provides similar functionality to the MU agent in the case of uncompressing data. The Mowser system also does not provide support for environment aware adaptation. For example, optimizations will not be tailored to current link quality or other characteristics of the environment.

The major advantages of the prototype implementation over the Mowser system are that it supports environment aware adaptation and compression of web pages before transmission across the wireless link. Furthermore, the prototype provides the usual benefit that the optimizations can be transferred to any foreign network (subject to the assumption that the mobility framework is supported) and it works within the existing Internet architecture. The Mowser system requires slight changes to the HTTP protocol and to end systems in general so that they are capable of providing different content type in response to the content negotiation performed. However, the Mowser system clearly wins in the optimizations that it performs and does raise interesting ideas about moving the optimizations out of the network and onto the end systems.

4.5.3 Teleweb

The implementation of the Teleweb system [Sch96] was primarily focused on overcoming the cost limitations of wireless networks by exposing the costs to the user. The Teleweb architecture consists of a single proxy on the mobile unit and as such it is not able to perform any optimizations to the data transmitted across the wireless link [Sch96]. The designers of Teleweb argued that link layer compression, web caching, *etc.* address the same issues as wired side proxies, and hence concentrated on the cost aspect of wireless networks. While it is true that link layer compression exists, the effectiveness of this compression is questionable when examined using monetary cost as a metric. Some wireless service providers charge by the uncompressed byte [Fra97], so a wired side component that compresses the data before it is transmitted will be beneficial to users of the service. Furthermore, Perkins [Per98] argued that compression can often be more effectively implemented at higher layers where the context of the data is known

Teleweb is effective in its area of specialization. By making costs visible through annotated HTML and postponing browser requests until certain criteria (e.g. budget constraints) are satisfied, Teleweb is able to present the user with a wide range of cost information and methods to minimize that cost [Sch96]. However, in cases where the user must perform web browsing the system is entirely ineffective. Knowledge of cost does nothing to alleviate the cost if the user is required to fetch certain pages at certain times. Waiting for a less costly connection may prove an unacceptable alternative in some circumstances. In these cases, the prototype system offers a clear advantage. Through the use of optimizations, the prototype is able to provide an actual savings in the number of bytes transmitted and overcome some of the limitations of wireless networks.

The principle advocated by the Teleweb designers "...that requiring dedicated servers reduces availability" [Sch96] is not a valid argument when applied to the prototype because the FU agent is dynamically dispatched onto each foreign network. The only real similarities between Teleweb and the prototype are that they both employ a cache on the mobile unit and both systems interact with the user through special, predefined web pages. It is important to note, however, that the optimizations employed by Teleweb and the prototype are not mutually exclusive. They could be combined in a hybrid system to offer both sets of functionality to the user.

4.5.4 WebExpress

The WebExpress system [Hou96] [Cha97] is particularly effective when applied in the transaction processing or commercial application domains. WebExpress employs a two proxy architecture similar to the one set up by the prototype. In addition, WebExpress supports the following optimizations: caching (on both the mobile and fixed units), differencing, header reduction and support for disconnected operation (through the queuing of web page requests) [Hou96]. The differencing optimization is aimed at repetitive transaction processing type applications. It is not quite as effective when presented with random *ad-hoc* web browsing where the user may load any arbitrary page. On the other hand, the optimizations performed by the prototype (such as compression and reducing the size of images) are more general. They do not offer any greater advantages in a particular domain but perform well in each domain. WebExpress has an advantage in transaction processing domains but falls short when confronted with other types of browsing. The optimizations it employs are tailored towards commercial forms-based applications.

Both implementations support caching but the caching performed in the prototype is more restrictive. It takes place only on the mobile unit because the FU agent may not have access to the filesystem on the fixed unit. Note, however, that a cache on the fixed unit is not strictly necessary here because the MU and FU agents communicate on a one to one basis in the prototype, which decreases the benefits of a fixed unit cache. Both WebExpress and the prototype communicate with a proprietary protocol (WebExpress removes some of the HTTP header information) and both attempt to overcome the problem of requiring a separate TCP/IP connection for each entity on a web page (WebExpress uses virtual sockets while the prototype independently fetches images). The most significant difference between the two implementations lies in their respective architectures (recall Section 3.4). Whereas the wired side component in WebExpress is fixed, the FU agent in the prototype can be dynamically dispatched into a foreign network thereby increasing the flexibility of the system.

4.5.5 Mobiscape

The Mobiscape system [Baq95] is one of the simplest mobile web browser systems. Similar to WebExpress, it uses a two proxy architecture. However, it performs only a few basic optimizations. It offers an enhanced caching policy that allows the user to control the cache content via a cache profile, in order to specify documents that the user always wants close at hand. Caches are resident on both the mobile and fixed units. It also performs data compression between the two proxies [Baq95]. Mobiscape suffers from the same inflexibility of WebExpress in its reliance on a wired side component. The prototype implementation overcomes this reliance and at the same time offers compression between the MU and FU agents. The prototype did not focus on caching a great deal other than requiring that a cache be present on the mobile unit. Consequently, the caching polices in the mobile unit are largely undefined. The caching enhancements provided by Mobiscape offer a demonstrated effective caching mechanism for mobile web browsing that could be integrated into the prototype (with the exception that the cache on the fixed unit would not be possible). Beyond this enhancement the Mobiscape system offers little to compare against.

4.5.6 Mowgli

Mowgli [Lil95] is the most similar to the prototype with respect to optimizations. Mowgli is another system built around a two proxy architecture. It employs several optimizations similar to those used by the prototype including inline image prefetching and caching. The proxy on the fixed unit parses the web page to extract image links, and loads the images before the proxy on the mobile unit requests them. In addition to these optimizations Liljeberg et al. [Lil95] proposed several other optimizations not currently implemented in the Mowgli system including compression of web pages, HTTP header compression, and tradeoffs in image fidelity. Taken together, these optimizations (proposed and implemented) match those provided by the prototype system described here. Furthermore, Liljeberg *et al.* [Lil95] described two other optimizations not considered in the prototype: Round trip transfer removal (actually implemented in Mowgli) and annotated HTML (proposed, but not implemented). Round trip transfer removal is similar to the notion of virtual sockets in WebExpress where several requests and responses may be communicated between the fixed and mobile unit over a single TCP/IP connection rather than using a separate connection for each HTTP request. Annotated HTML involves altering the HTML on the fixed unit before it is transmitted to the mobile unit [Lil95]. Annotations are added to links to expose certain properties that may be useful to mobile users. For example, a link in a web page may be annotated to provide information such as the size of the linked object or the availability of the linked object in the cache. The only optimization the prototype supports that Liljeberg *et al.* [Lil95] do not mention is a user profile to select the desired type of information.

When comparing the implementation of Mowgli (which lacks certain proposed optimizations) with the prototype system discussed here, the prototype offers a wider range of functioning optimizations. However, if the entire set of optimizations identified by Liljeberg *et al.* [Lil95] were incorporated into the Mowgli system, it would be more effective than the prototype (with respect to optimizations). In general, both Mowgli and the prototype provide similar optimizations and are effective within the domain of *ad-hoc* web browsing because neither employs optimizations tailored to a particular niche of web browsing such as transaction processing. The major difference between them lies in the underlying architecture. As with WebExpress [Hou96] the Mowgli system relies on a wired side component that limits its flexibility.

Chapter 5

TESTING/ASSESSMENT

Following the development of the prototype, a series of tests were performed to demonstrate the correctness of the system and to uncover any hidden problems that had arisen during the implementation. Testing was done to ensue that the prototype operates in the manner specified in Section 3.3. The tests only verify the behavior of the system, they do not assess performance (which depends on the optimizing operations employed). This chapter describes the testing process and the system tuning that took place as a result of the initial tests.

5.1 Testing To Demonstrate Correctness

The prototype was developed on a wired Ethernet network consisting of four Windows NT machines running at 300 MHz with 128 Megs of RAM. During development several preliminary tests were conducted to ensure that the system was stable and operated correctly. Upon completion of the system development, the entire prototype was ported over to the wireless testbed network (Figure 4.1) to test the system within its actual operating environment. To this end, a comprehensive test plan (see Appendix C) was drafted that was designed to isolate and test specific aspects of the system. The system was required to pass each of the tests in this plan. This section presents the details of the tests that were employed.

Testing of the system was broken down into several different stages. Initially a test suite of web pages was identified. The pages in the test suite were chosen for either the specific content that they contained or the presumed frequency with which they are used in regular web browsing to present the system with a diverse selection of commonly occurring pages and to isolate certain HTML features. Table 5.1 provides the URL for each of the test pages along with a rational for choosing the page.

Table 5.1 - Web Page Test Suite

	Web Page	**URL**	**Reason**
1	University of Manitoba Homepage	http://www.umanitoba.ca/	Common page visited by a large number of students
2	University of Manitoba Dept. of Computer Science Page	http://www.cs.umanitoba.ca/	Common page, mostly text based
3	Dr. John Bate's Java test page	http://www.cs.umanitoba.ca/~bate/JavaEgs/Face.html	Contains an example of a Java applet
4	Dr. Randal Peter's Homepage	http://www.cs.umanitoba.ca/~randal	Good mix of both text and images (gif and jpeg)
5	Dr. Peter Graham's Homepage	http://www.cs.umanitoba.ca/~pgraham	Contains a large gif image (useful to test image compression)
6	Gilbert Detillieux's poutine page	http://www.cs.umanitoba.ca/~gedetil/poutine.html	Contains an audio file
7	Altavista search engine	http://www.altavista.com	Common search engine
8	Infoseek search engine	http://www.infoseek.com/	Common search engine
9	Metacrawler search engine	http://www.metacrawler.com/	Common search engine
10	Amazon online bookstore	http://www.amazon.com	Popular online e-commerce site. Contains a forms based search.
11	ACM digital library search page	http://www.acm.org/dl/newsearch.html	Contains a forms based search
12	City of Winnipeg Homepage	http://www.city.winnipeg.mb.ca/city/	Contains a large number of images

	Web Page	URL	Reason
13	Government of Manitoba Homepage	http://www.gov.mb.ca/cgi-bin/choose_home.pl	Contains a large number of images
14	Xing Chen's Homepage	http://home.cc.umanitoba.ca/~umchenxz/	Contains a mixture of images and Java
15	Flora Wang's Homepage	http://home.cc.umanitoba.ca/~umwangw/home.html	Contains an HTML frame

Once the test suite was determined, a series of initial tests were conducted that focused on the system as a whole rather than individual system components. These initial tests were concerned with the ordinary operation of the system and consisted of using the system for random *ad-hoc* web browsing for an extended period of time (between 20 and 30 minutes). During this time, regular web browsing was performed on the mobile unit with the system operating in the background. Common web pages were loaded (including various search engines, often used in *ad-hoc* web browsing), and links on pages were followed. Testing of this nature was performed for the following user profile settings: text and images, text only, and links and images only. The user profile settings were changed at arbitrary points in time during the browsing session. These tests were also used to verify the compression of the HTML pages. The MU agent log file was checked after the testing to ensure that the sizes of the pages received were less than the actual page sizes (typically the web page size was compressed by around 50%)

After being satisfied that ordinary system operation functioned correctly, the individual profile settings were tested in more depth. A particular profile setting (e.g. text only) was selected and a sequence of six predefined pages (test pages 1, 2, 4, 5, 12 and 13 chosen from the test suite for their combination of images and text) was loaded to verify that the profile was correctly filtering the information being displayed.

The next phase of testing relied on the event agent simulator component of the MU agent to test the system operation under varying communication's link quality. The two link qualities tested were poor and no connection because the good and acceptable link qualities produce the same behavior of the system as in the general case (described in the previous two paragraphs). A poor link quality was simulated and tested using a scenario file with the single entry (poor, 1000). Under this scenario the image compression optimization of the profile was tested. A sequence of pages was loaded that contained both gif and jpeg images and the images were examined to verify that compression had taken place (this included visual verification and checking the size of the transmitted images logged in the MU agent log file). A no connection link quality was tested with a scenario file containing the entry (no connection, 1000). In this case, the system acted appropriately by displaying a message box to the user indicating that there was no network connection. More detailed scenario files were also constructed to observe the system behavior during changing link qualities. This involved scenarios where the quality started out good and progressed to poor or no connection before returning to good again. By using a no connection entry in a scenario file it was possible to simulate disconnections and ensure that the system handled them appropriately.

Once testing of the system with the event agent simulator was completed the system was tested using the actual WaveLAN link quality monitor to ensure that the system could adapt to the changing link quality when it was monitored in real time. To achieve the desired link qualities the mobile unit was moved during these tests to areas of the building (housing the testbed) in which the wireless network signal strength is poor. The same tests as above were then performed to ensure that the behavior of the system using the WaveLAN link monitor was consistent with the behavior observed when using the event agent simulator.

The next stage of the testing process involved verifying that the system was capable of retrieving a diverse range of media types contained in web pages. The following media types (in addition to text and images) were successfully loaded through the system: audio files, pdf and postscript files, Java class files and program installation files.

After testing the system with various media types, different aspects of HTML and the HTTP protocol were examined. With respect to HTML, the system was tested on web pages that contained frames and forms. Frames were tested by loading a web page containing frames (page 15 from the test suite in Table 5.1). Forms were tested by loading the amazon.com and ACM digital library web pages (pages 10 and 11) and performing searches for books and articles, respectively. Through testing forms processing, the handling of HTTP POST requests by the system was also tested because the forms on these pages use an HTTP POST method to submit data to a server.

The focus of the testing was then shifted to the behavior of the FU agent. In particular, three operations involving the FU agent were examined. First, the basic FU agent dispatch process was tested to ensure that the FU agent always starts properly upon the first request the MU agent receives from the web browser. Secondly, the behavior of the system in response to FU agent errors was verified. After dispatching a FU agent to the agent system on the fixed network and using it for some period of time, the FU agent process was manually killed using features provided by the Tahiti aglets server. Browsing then continued on the mobile unit to ensure that the system is robust against such errors. As expected, the error was detected and a new FU agent was dispatched. Following the initial testing of a FU agent error the process of killing the FU agent was repeated 3 consecutive times. This was done to check that once the number of errors has exceeded the predefined tolerance level, the MU agent would no longer dispatch the FU agent. In this case the MU agent resorted to using a direct connection, which was the expected behavior. The third aspect of the FU agent to be tested was FU agent behavior when multiple FU agents are executing simultaneously in the agent system. To simulate this, after a FU agent was dispatched, the link quality was set to no connection for a brief time period. When the MU agent was reconnected again, it dispatched a new FU agent into the foreign network and two FU agents executed together in the agent system. As expected, the new FU agent handled the processing of the MU agent requests. The original FU agent timed out and halted execution.

Tests were then performed to evaluate the various timeouts employed by the system. In particular, there are two places that timeouts may occur: on the FU agent, when a request has not been received for an extended period of time and on the MU agent while waiting for a FU agent to be dispatched. In the case of the FU agent timeout, the number of timeouts that occurred during regular browsing sessions was noted to make sure that it is close (preferably equal) to zero. Regarding the MU agent timeouts, the FU agent dispatch time was examined to ensure that the MU agent timeout value is slightly larger. Furthermore, the MU agent timeout was allowed to expire (in which case a direct connection is used) to make sure that the timeout duration is not prohibitive for users who are waiting for a page to be loaded.

The final component of the system to be examined was the direct connection module. The service agent executing on the fixed network was turned off so the MU agent would not be able to locate an agent system that could host the FU agent. As expected, the MU agent resorted to using a direct connection to the web server in this case to allow web browsing to continue on the foreign network (even though the network does not support the FU agent). The operation of the direct connection itself was tested with the various media types (audio files, pdf files, etc), web pages containing frames, web pages containing forms and all the pages in the test suite to ensure that, from a user's perspective, the operation of the system is the same regardless of whether a direct connection or the FU agent is used.

5.2 Tuning Based on Initial Observations of System Testing

Following the initial selection of tests outlined in the test plan and described above some system parameters and behavior were adjusted to overcome problems that were uncovered. Through the testing, it was discovered that the timeout values used for both the MU agent (while waiting for dispatch of a FU agent) and FU agent (the amount of time the FU agent will exist in the absence of any requests from the MU agent) were too short. In the case of the MU agent this resulted in direct connections being used instead of transmitting requests to the FU agent because the MU agent threads waiting for the FU agent dispatch timed out and assumed (incorrectly) that the FU agent could not be dispatched (when, in fact, the FU agent dispatch was just taking longer than expected). Regarding the FU agent, timeouts were occurring too frequently when waiting for web page requests. Since users spend some time reading the web pages they load, the timeout period used initially was far too short. It was appropriate for testing because it forced timeouts to occur within a reasonable amount of time. Unfortunately, once displayed, in the time it takes a user to read a web page the FU agent would timeout. Both of these problems were resolved by using longer values for the timeouts.

With respect to the timeouts, it was also determined that if a handshake is received by the MU agent from a FU agent while the FU agent manager (on the MU agent) is in a not allowed state (which indicates that the MU agent could not properly dispatch a FU agent onto the foreign network), then the FU agent manager should move to an active state. Previously, once a not allowed state was encountered, the manager could only move back to the initial inactive state. However it was noted during testing that in the event of a timeout on the MU agent where the timeout was caused by the timeout value being too short rather than an actual failure to dispatch the FU agent, the FU agent manager will move into a not allowed state prematurely. By enabling a move from a not allowed state to an active state on receipt of a handshake, the system will still be able to make use of the FU agent (and will use direct connections in the interim before the handshake is received). This has the added benefit that the time to transmit a request for a web page is always bounded by the MU agent timeout value. Furthermore, this change was necessary since it is very hard to predict the optimum timeout value for all possible foreign networks that have varying bandwidth availability. It is better to set a reasonable timeout value and to deal with timeouts as they occur by using direct connections. When the FU agent is set up (as signified by the handshake), the MU agent can switch from direct connections to the FU agent.

When examining the WaveLAN link test log file (used for real time monitoring of the communications link quality) it was discovered that some of the entries written to the log file (in particular the disconnection entry) were not in the format expected by the Event agent module. The Event agent was modified so that it uses the same format as the log file to correctly detect changes in the link quality.

Two problems were noted with respect to initialization of the MU agent. First, an Event agent initialization file was included with the setup files, but never used. This file was deleted because it contains no useful information. Secondly, the location of the FU agent code (the FUAgent.jar file) was hard coded into the MU agent. This location was moved from the system code to the MU agent initialization file to support more flexible relocation of the FUAgent.jar file.

There were also a couple of problems noted with the cache on the MU agent. A runtime error occurred when trying to clear an empty cache and, in some cases, images written to the cache were unreadable. The first problem was fixed by checking whether or not the cache is empty before clearing it. The second problem was due to the way in which images were written to the cache. The original implementation converted the images to a Java string and then wrote the string to a file. This conversion process caused problems with the encoding of the images. To solve these problems, the raw byte array format of the images was written to the cache file instead of using the string format.

Chapter 6

CONCLUSIONS

6.1 Summary and Contributions

The primary result of this work is a new architecture designed to support efficient web browsing at the application layer in a wireless mobile computing environment. Mobile computing environments present several challenges to traditional networked applications arising from the limitations inherent in wireless networks and the inability of conventional network protocols to cope with these limitations. Previous research in this area has identified the client/intercept/server architecture as an effective solution to enable optimizations of the application layer communications that take place over the wireless link. Two proxies characterize this architecture: one proxy which executes on the mobile unit and the other proxy which executes on the fixed unit. Together these proxies encapsulate communications over the wireless link and control the type and format of the application layer information flowing across the link. The major disadvantage in such an architecture is the reliance upon a wired side component. Requiring the existence of such a component on foreign networks hinders mobility. The optimizations performed by systems employing this architecture will not be accessible on any network that does not have the wired side component installed. The new architecture described in this thesis uses mobile agent technologies to dynamically dispatch the wired side part of the client/intercept/server architecture into the foreign network with which the mobile unit is communicating. The specific contributions of this thesis are the following:

1. A new architecture is described that allows a mobile unit to dynamically deploy a client/intercept/server based system onto foreign networks with which it

communicates. The new architecture models the wired side component in the client/intercept/server architecture as a mobile agent that travels to a fixed unit on the foreign network to establish a proprietary application layer communications channel with the mobile unit. This dispatch of the agent replaces the reliance on a wired side component with the reliance on the framework described in Section 3.2.

2. The architecture described here does not require any changes to the existing Internet protocols. It can be entirely implemented using existing protocols and software as is demonstrated through the creation of a prototype system based on the architecture.
3. A framework is identified that enables mobile units to discover and use mobile agent systems on foreign networks.
4. Mobile agent systems on foreign networks are modeled as a service that the foreign network provides, similar to other services such as printers.
5. Service location protocol (SLP) service templates are created for agent system service types, to enable the discovery of agent systems via SLP.

The architecture described in this thesis is flexible. The mobility framework identifies the services that could be provided by the foreign network to allow mobile users to discover and use mobile agent systems running on the foreign network. The framework does not specify how these services are to be implemented. The prototype demonstrated one specific implementation to illustrate the feasibility of the architecture. This implementation relied on DHCP, SLP and the Aglets agent system. Other approaches are equally acceptable. A requirement does exist that an implementation of the architecture must be compatible with the implementation of the mobility framework.

Similarly, there is a great deal of flexibility in the optimizations that can be employed in conjunction with the mobile web browser support system. Optimizations are by no means limited to those used in the prototype. Rather the architecture suggests a general way to set up a system that encapsulates application layer communications across a wireless link. Once the client/intercept/server architecture is established an implementation is free to employ any number of optimizations including, but not limited to, those discussed in the prototype implementation and those used by the other related systems described in Chapter 2. However, the optimizations are limited to the application layer.

The benefits offered by a system employing the architecture described here depend heavily on the optimizations that are employed. Optimizations that reduce the number of bytes transmitted across the wireless link (such as the compression of web pages used in the prototype) could be translated into cost savings for the mobile user on wireless networks that charge by the number of bytes transmitted. Other optimizations that reduce the connection time could offer lower cost on networks that charge by the amount of time a user remains connected. In either case tangible benefits could be realized for the user in terms of monetary savings by tailoring the optimizations to the cost model employed by the foreign network. From the point of view of wireless service providers, optimizations that reduce the number of bytes transmitted across the wireless link would be beneficial as they reduce the load on the network. Note that this thesis does not claim any specific performance benefits as no performance analysis was conducted. The goal of this thesis was to deploy a system that enables application layer optimizations, not to prove the effectiveness of those optimizations. The benefits mentioned previously are examples of results that may be attained provided sufficient optimizations were implemented.

Another result of this thesis is that it demonstrates the effectiveness of mobile agents in mobile computing environments. Previous research on mobile agents in mobile computing environments has focused to a large extent on using mobile agents to offload computations to a fixed network or to support disconnected operation by continuing to execute on the fixed unit while the mobile unit is disconnected. This thesis shows how mobile agents can be used to encapsulate a proprietary protocol.

Finally the mobility framework developed as a part of the new architecture offers potential benefits extending beyond the deployment of a client/intercept/server architecture onto foreign networks. The framework provides a means by which mobile users can discover and use mobile agent services on a foreign network that supports the framework. This capability provides a means of injecting user specific application layer services into networks with which the mobile user communicates. It is difficult for a foreign network to anticipate the needs of the diverse range of mobile users with which it communicates. Different mobile users require different services from the network. Mobile agents could offer a mechanism by which the mobile user can place their own value added services into the network. The service provider only needs to support an agent system, rather than supporting the services required by each user. Users may discover the agent system and transmit their own services (implemented as mobile agents) into the network.

6.2 Future Work

The research presented in this thesis opens up several avenues of additional work that could be pursued. The use of mobile agents to encapsulate proprietary protocols has been identified in earlier research [Lan97], but has not been thoroughly examined. This application of mobile agents seems particularly relevant to the low bandwidth networks presented by wireless mobile computing environments where proprietary protocols can potentially make more efficient use of the bandwidth. The architecture presented here employed mobile agents in this fashion but further research could identify whether such a technique is generally applicable, especially in the context of mobile computing.

Furthermore, the model of a mobile agent system as a service offered by foreign networks needs to be refined. A set of attributes should be identified that would provide information about the agent system including agent system type, security privileges, and other facts that could aid in using the system. This thesis presented only a minimal specification of the agent system service designed to assist in the location of the agent system on a foreign network. Although appropriate for the architecture presented here this specification should be enhanced to provide information that would be required by other applications searching for a mobile agent system. Further work must also be done on agent systems in general to achieve compatibility between different agent system types. Agent system interoperability would make it easier for networks to provide agent services since the networks could offer a single agent system as a service that is capable of hosting a variety of different agent types.

With respect to mobile web browser support systems more research needs to be done to identify the best set of optimizations to perform. There is no agreement on how to optimize communications across a wireless link or where to apply the optimizations (proxy based or end system). The prototype system presented here and the related systems that are discussed in Chapter 2 each perform a unique set of optimizations. Although sometimes similar, there is no consensus on which optimizations are ideal and to the best knowledge of the author no rigorous comparison has been performed between the various systems. Such a comparison would be helpful in identifying those optimizations that offer the best performance.

An in depth performance analysis should also be performed to gain an understanding of the overall effectiveness of the architecture in terms of performance. The analysis should clearly measure the overhead associated with the architecture described here (with respect to dispatch of the FU agent and the indirection of requests which must pass through two proxies) versus the benefits yielded by the optimizations employed. Furthermore the analysis should consider the scalability of the approach taken.

Finally further work could be done to extend the architecture presented in this thesis to support several clients (MU agents) for a single FU agent. The architecture was developed such that the MU agent and FU agent operate on a one to one basis. Extending this so that one FU agent communicates with many MU agents would be beneficial in those cases where the FU agent is common enough that many MU agents want to access the same FU agent services at the same location.

Appendix A

Agent System Service Templates

A.1 Overview of SLP Service Templates

An SLP service template describes a particular service type. Service templates are used to define the attributes associated with the service type. These attributes allow clients querying a service to distinguish between different instances of the same service type. For example, a template for a printer service type may define an attribute called color. Instances of the service type printer can provide a value for the color attribute depending on whether or not they support color printing. Service templates also describe the syntax of service: URLs for instances of the service type. A service: URL identifies a particular resource offering a service (for example, a printer) and specifies unambiguously how to access the service. Service templates are composed of five distinct items:

1. Template Type – The template type provides the name of the service type that the template describes.
2. Version – The version specifies when the template was created. All templates start with a version number of 0.0. After standardization they are assigned the version number 1.0. Changes in the template cause the version number to be incremented.
3. Description – The description item indicates what the service described by the template does.
4. URL Syntax – The URL syntax specifies how to interpret the URL path of a service: URL. The URL path is the part of the service: URL following the address of the resource. For example, consider the service: URL service:x-agentsystem.local:atp://mira.cc.umanitoba.ca/myPlace which is based

on the service templates provided in this appendix. The URL path of this service: URL is myPlace (i.e. the part of the service: URL that follows the address of the host offering the service). According to the template for the x-agentsystem.local:atp service this URL path is to be interpreted as a context name. A context name is the name of a specific execution environment within an Aglets mobile agent system. Recall from the discussion on mobile agents in Chapter 2 that an agent system can have more than one execution environment (place). The URL syntax described in this template allows a service: URL to refer to a specific place within an agent system. In a service template the URL syntax item contains the grammar specifying the syntax of the URL path. For the context name example the grammar indicates that the context name is composed of any sequence of alphanumeric characters.

5. Service Attributes – Service attributes describe a set of attributes whose values can be used to differentiate between two instances of the same service type.

Refer to Kempf [Kem99b] and Guttman [Gut99] for a more detailed description of the syntax for service templates and service: URLs.

A.2 Agent System Service Templates

This section contains the service templates for two new service types that are used for discovery of a mobile agent system service via SLP. The first service template defines the x-agentsystem.local abstract service type that describes a general agent system service. Abstract service types aggregate several services that use different protocols to provide the same service. An abstract service type is appropriate for the base agent system service because there is no agreed upon protocol for communicating between different agent system types. Agent systems are not currently interoperable and they use different proprietary protocols for communications.

The second service template defines the x-agentsystem.local:atp concrete service type. Concrete service types are used to specify a single protocol within an abstract service type. The x-agent system.local:atp concrete service type is an instance of the x-agentsystem.local abstract service type. It specifies how clients may access an agent system service using the agent transfer protocol (ATP). Currently only IBM's Aglets agent system employs this protocol.

In both cases, the x in the service type name indicates that the service type is experimental. Furthermore the local extension to the service type names shows that they are defined by a local naming authority (i.e. the names are not standardized). No attributes are currently defined for either of the service types. The templates are provided here to formalize the service type names and URL path syntax. For the purposes of the prototype implemented in this thesis, locating the service based on the service type name alone is sufficient. The only agent system using the ATP protocol is IBM's Aglets agent system. Hence any agent system that advertises a service type specifying the ATP protocol (for example, x-agentsystem.local:atp) will be compatible with the Aglets system. No other attributes are required to determine compatibility. As agent systems progress towards interoperability other attributes would be beneficial since the access protocol alone may not be capable of identifying a particular agent system type.

```
------------------------- template begins here ------------------------
template-type=x-agentsystem.local

template-version=0.0

template-description=
 This is an experimental abstract service type. The purpose of the
 x-agentsystem service is to organize into a single category all
 agent systems that are capable of hosting mobile agents.

template-url-syntax=
 url-path=  ; This abstract type does not define a url-path
-------------------------- template ends here -------------------------
```

```
------------------------- template begins here ------------------------
template-type=x-agentsystem.local:atp

template-version=0.0

template-description=
 This concrete service type defines the url syntax for agent systems
 that use the Agent Transfer Protocol (ATP). This is the protocol
 employed by the Aglets agent system. An agent system specified by this
 type is assumed to be capable of hosting aglets (mobile agents defined
 for IBM's Aglets agent system). Mobile agents and messages may be
 transmitted to the agent system via the ATP protocol. This service
 type is a concrete instance of the x-agentsystem abstract service
 type.

template-url-syntax=
 url-path= [contextname]

 contextname= 1*alphanum

 alphanum= alpha / digit

 alpha= "a" / "b" / "c" / "d" / "e" / "f" / "g" / "h"/ "i" / "j" /
        "k" / "l" / "m" / "n" / "o" / "p" / "q" / "r" / "s" / "t" /
        "u" / "v" / "w" / "x" / "y" / "z" /
        "A" / "B" / "C" / "D" / "E" / "F" / "G"/ "H" / "I" / "J" /
        "K" / "L" / "M" / "N" / "O" / "P" / "Q" / "R" / "S" / "T" /
        "U" / "V" / "W" / "X" / "Y" / "Z"

 digit= "0" / "1" / "2" / "3" / "4" / "5" / "6" / "7" / "8" / "9"
-------------------------- template ends here -------------------------
```

Appendix B

Proprietary Protocol

The proprietary protocol is an application layer protocol used for communications between the MU agent and the FU agent. The protocol uses TCP as the transport layer protocol and supports two major types of messages: Requests and Responses. Appendix B describes these message types in detail.

Note that in all messages, a newline character always follows variable length strings.

B.1 Requests

Requests are messages that are transmitted from the MU agent to the FU agent. The destination port on the FU agent is determined dynamically and communicated to the MU agent by a FU Agent Started response message (see section 2.1 below and section 4.3.2.2, FU agent operation). The MU agent may use any ephemeral port as the source port. There are four types of request messages that are supported: Profile, Get, Post and Link Status Changed.

B.1.1 Profile Messages

Profile messages indicate that the user profile stored on the mobile unit has changed. They contain the new profile settings, so that the FU agent can update its stored copy of the user profile (and adjust its processing accordingly).

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Header = 1    | Profile Value |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

New User Profile Settings		Profile Value
Text	Images	
Yes	Yes	1
Yes	No	2
Yes	Compressed	3
Links Only	Yes	4
Links Only	Compressed	5

B.1.2 Get Messages

Get messages contain a reformatted HTTP GET request from the web browser for a specific HTTP entity (such as a web page). The URL of the requested entity has been extracted from the HTTP GET request and is contained in the proprietary protocol Get message.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Header = 2    | Requested URL (Variable length string)        \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

B.1.3 Post Messages

Post messages are used to tunnel an HTTP POST request to the FU agent. They contain the original HTTP POST request that was transmitted to the MU agent from the web browser encapsulated in a proprietary protocol message. The HTTP Post request field (shown below) contains the original HTTP POST request transmitted by the web browser.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Header = 8    | Requested URL (Variable length string)        \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       HTTP Post Request Length                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       HTTP Post Request                       \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

B.1.4 Link Status Changed Messages

Link status changed messages are transmitted to the FU agent to provide an update on the current quality of the communications link. They are only transmitted when a change in the link quality is detected and contain the new link quality.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Header = 16   | Link Quality  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Link Quality	Value
Good	0
Acceptable	1
Poor	2

B.2 Responses

Responses are messages that are transmitted from the FU agent to the MU agent. The destination port on the MU agent is always 9000. The FU agent uses an ephemeral port as the source port. There are seven types of response messages: FU agent started, FU agent shutdown, HTTP entity, URL changed, Images, Post response and Error.

B.2.1 FU Agent Started Messages

The FU agent transmits a FU agent started message immediately upon execution in an agent system. The purpose of the message is to indicate to the MU agent that the FU agent has been successfully dispatched, and to inform the MU agent how to communicate with the FU agent. FU agent started messages contain the unique identifier (xid) of the FU agent (which is included so that the MU agent can determine which FU agent has transmitted the message, in case more that one FU agent has been dispatched), and the port on which the FU agent will be listening for requests. The port transmitted in the FU agent started message is chosen at random by the FU agent (so as to minimize conflicts with other FU agents executing in the same agent system). The MU agent should use the port as the destination port for all messages it sends to the FU agent.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Header = 32   |                    xid                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| xid, contd.   |                    Server Port                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| port, contd.  |
+-+-+-+-+-+-+-+-+
```

B.2.2 FU Agent Shutdown Messages

The FU agent transmits a FU agent shutdown message immediately before it halts execution. This usually occurs in response to a timeout on the FU agent (which happens when the FU agent has not received a request from a MU agent for some predetermined amount of time). A FU agent shutdown message indicates to the MU agent that the FU agent is no longer active. The message contains the unique identifier (xid) of the FU agent, so that the MU agent can determine whether or not it was communicating with the FU agent. In the case where the MU agent was not communicating with the FU agent that terminated (i.e. the xid stored in the MU agent does not match the xid transmitted in the message), the shutdown message is ignored. Otherwise, the MU agent notes that the FU agent has shutdown, so that it may dispatch another FU agent if the services provided by the FU agent are still required. FU agent shutdown messages are only transmitted when the FU agent is gracefully terminated. This message will not be transmitted in cases where the FU agent is terminated due to some error or unexpected event (such as the agent system on which the FU agent is executing crashes).

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Header = 64   |                    xid                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| xid, contd.   |
+-+-+-+-+-+-+-+-+
```

B.2.3 HTTP Entity Messages

HTTP Entity messages are the responses transmitted to get request messages. They contain the URL of the requested entity (as sent in the get request message) so that the MU agent is able to match up the response with the request that generated it. They also contain the requested entity, which the FU agent has retrieved from a web server, the content type of the entity (since an entity could be any number of different types including HTML pages, Java class files, pdf files, etc) and a time to live value for the entity (the amount of time that the entity may remain in a cache before being refreshed). The entity contained in the message may or may not be compressed. The entity size field always gives the size of the entity data (if the entity is compressed then this field gives the compressed size). If the entity is compressed, then the uncompressed size field gives the uncompressed size of the entity. Otherwise the uncompressed size field is set to zero.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Header = 1    | Requested URL (variable length string)        \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Content Type (variable length string)  \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Time To Live                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Time To Live, contd.                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Uncompressed Size                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Entity Size                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Entity (bytes)                         \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

B.2.4 URL Changed Messages

URL changed messages are nearly identical to HTTP entity messages. They are transmitted in response to get messages where the requested URL resulted in an HTTP redirect message (when the entity was retrieved from the web server) and the URL contained in the redirect message was used to retrieve the requested entity. URL changed messages are used to transmit the requested entity back to the MU agent, while at the same time communicating that a redirect has taken place. The redirection is handled entirely by the FU agent, to avoid passing the redirect messages across the wireless link. URL changed messages contain the same information as HTTP entity messages, plus the URL that was actually used to retrieve the entity. Effectively they combine together an HTTP redirect message and a get response message. The requested URL field contains the URL the MU agent used to request the entity. The actual URL field contains the URL that was used to retrieve the entity from the web server (i.e. the redirect URL). All other fields are the same as for an HTTP Entity message.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Header = 8    | Requested URL (variable length string)        \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             Actual URL (variable length string)     \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             Content Type (variable length string)   \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             Time To Live                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             Time To Live, contd.              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             Uncompressed Size                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             Entity Size                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             Entity (bytes)                    \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

B.2.5 Images Messages

Images messages contain all of the images referenced on a particular web page. The FU agent transmits an images message after it has fetched all the images for a web page that was requested by the MU agent. An images message contains the URL of the requested web page (i.e. the page for which the images were retrieved) so that the MU agent is able to match up the images with a particular web page request.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Header = 2    | Requested URL (variable length string)        \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             Number of Images (N)              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             N Images                          \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Each image contained in the images response message is described by an image URL, the content type (i.e. type of image), the size of the image and the image data itself.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Image URL (variable length string)      \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Content Type (variable length string)   \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Size                                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Image Data                              \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

B.2.6 Post Response Messages

Post response messages contain entities retrieved from post request messages. They are identical in format to HTTP Entity messages except that the header value is 16 instead of 1. Post response messages are differentiated from HTTP Entity messages because the MU agent handles them differently. The MU agent does not cache responses to post requests, but does cache the entities contained in regular HTTP Entity response messages.

B.2.7 Error Messages

The FU agent transmits error messages when an error occurs processing a request from the MU agent. Error messages contain the URL of the entity that was being retrieved when the error occurred and a string describing the error.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Header = 4    | Requested URL (variable length string)        \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|               Error Message (variable length string)          \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Appendix C

Wireless Network Test Plan

This appendix contains the test plan followed to evaluate the prototype system.

1. System installation and configuration ___
    - Set up the prototype on the wireless test network and adjust configuration parameters in the .ini files.
    - Install the Aglets Agent system on the fixed unit (peppertree) and set up the environment variables.
    - Install the Agent System service agent on the fixed unit and adjust its initialization file.

2. General system testing ___
    - Perform web browsing with the system operating in the background. Visit the various pages in the test suite and follow links on the pages.
    - Ensure that the system functions correctly with several common search engines including Altavista, Infoseek and Metacrawler
    - Test the text only, images only and text and images user profile settings.
    - Check the log file generated by the MU agent to make sure all the transmissions between the MU agent and the FU agent are being logged appropriately
    - Verify that compression of text files is taking place (by examining the size of the web pages transmitted, as recorded in the log file).

3. User profile testing ___
    - Identify six test pages from the test suite that will be used to test the profile settings. The pages selected were:
      http://www.umanitoba.ca/
      http://www.cs.umanitoba.ca/
      http://www.cs.umanitoba.ca/~randal
      http://www.cs.umanitoba.ca/~pgraham
      http://www.city.winnipeg.mb.ca/city/
      http://www.gov.mb.ca/cgi-bin/choose_home.pl
    - Start the system and choose a profile (one of text only, images only or text and images)
    - Load the six test pages using that profile. Ensure that the displayed pages are consistent with the profile settings selected.

4. Environment aware adaptation testing using the event agent simulator ___
   - Create a scenario file that will simulate a poor link quality. Use the profile to select the image optimization option (combined with both the text = yes option and the text = links only option) and load the six test pages identified above in step 3.
   - Visually verify that the images on the pages have been compressed in size. Also examine the MU agent log file to ensure that the size of the transmitted images is less than the size of the actual images.
   - Create another scenario file to simulate disconnected operation.
   - Verify that the system still loads cached pages when the mobile unit is disconnected. Also check that when a page is requested which is not in the cache, the MU agent displays a message box to the user indicating that there is currently no network connection.
   - Finally create a number of scenario files that simulate a varying link quality. Use at least one file that simulates a disconnection (via a no connection entry in the file) during the browsing session and another that simulates a change from a good link quality to a poor link quality in order to verify the system operation in response to dynamically changing network conditions.

5. Test the environment aware adaptation using the actual WaveLAN link monitor ___
   - Move the mobile unit to a location where the link quality is poor and ensure that the image compression optimization works appropriately (again by loading the same 6 pages identified in step 3)
   - Move the mobile unit to a location where there is no network connection and check that a no connection message box is displayed.
   - Generally repeat the checks performed in step 4 (with the exception of the complex link quality simulation) in order to ensure consistent system behavior regardless of whether a real time link monitor or a simulation is being used.

6. Test the system with a number of diverse media types ___
   - Sound files (use the poutine audio file on the page http://www.cs.umanitoba.ca/~gedetil/poutine.html)
   - Pdf files (download a pdf file from the ACM digital library, http://www.acm.org/dl/newsearch.html)
   - Java programs (load http://www.cs.umanitoba.ca/~bate/JavaEgs/Face.html and http://home.cc.umanitoba.ca/~umchenxz/ and check that the Java programs contained on these pages execute properly)
   - General file transfer (Download the Adobe Acrobat Reader program from the Adobe website and install it on the mobile unit)

7. Test the system with HTML frames ___
   - Load the page http://home.cc.umanitoba.ca/~umwangw/home.html which contains a frame to make sure that frames are handled properly by the system

8. Test the system with HTML forms ___
    - Ensure that forms processing (the treatment of HTTP POST requests by the system) is still handled correctly. Load the acm digital library page (http://www.acm.org/dl/newsearch.html) and perform a number of searches. Then load the amazon.com online bookstore web page (http://www.amazon.com) and search on different book titles. Make sure that the format and content of the results returned is identical to that returned in a normal web browsing system without the use of the prototype system.

9. Test the FU agent behavior ___
    - Verify that the FU agent is always dispatched correctly upon receipt of the first web page request from the MU agent.
    - Check that 2 FU agents executing simultaneously within a single agent system does not cause a problem (Can artificially cause this to happen by using the event agent simulator with a scenario file that sets the link quality to good, then no connection and then good again. Attempting to load a page during the disconnection will reset the FU agent manager, resulting in a second FU agent dispatch when the next web page request is received and the link quality is good again).
    - Check that after 3 FU agent errors, the MU agent will give up using the FU agent and employ a direct connection to satisfy requests. FU agent errors can be simulated by manually killing the FU agent process on the fixed unit after the MU agent has dispatched it.

10. Test the FU agent and MU agent timeout values ___
    - Verify that the chosen timeout values on the MU agent and the FU agent are not too short or too long.

11. Check SLP ___
    - Verify that the Agent System service agent correctly provides the location of the fixed unit hosting the agent system (note that this has been implicitly checked while verifying the proper dispatch of the FU agent. It is included as a separate point for completeness.)
    - Check that if no agent system is present on the fixed network (i.e. SLP service discovery by the MU agent fails), the MU agent will use a direct connection to satisfy the web page requests. The absence of an agent system of the foreign network can be simulated by turning off the service agent that advertises the agent system before running the MU agent on the mobile unit.

12. Test the direct connection ___
    - Verify that the direct connection operates in the same way as a normal Internet connection would with respect to web browsing. Load a selection of pages from the test suite and ensure that they are displayed correctly.
    - Test the direct connection with the media types identified in step 6.
    - Test the direct connection under disconnected operation to make sure that it is robust against failure (i.e. move to a location where there is no network

connection available, and make sure that the attempted use of a direct connection does not crash the system).

References

[Air99] AirPort Fact Sheet, 1999, http://www.apple.com/airport/

[Ami98] Elan Amir, Steven McCanne and Randy Katz, An Active Service Framework and its Application to Real-time Multimedia Transcoding, SIGCOMM '98, Vancouver, BC, September 1998, pp. 178-189

[Baq95] Carlos Baquero, Victor Fonte, Francisco Moura and Rui Oliveira, MobiScape: WWW Browsing under Disconnected and Semi-Connected Operation, Proceedings Portuguese WWW National Conference, Braga, July, 1995, http://gsd.di.uminho.pt/

[Bec99] James Beck, Alain Gefflaut and Nayeem Islam, MOCA: A Service Framework for Mobile Computing Devices, MobiDE, Seattle WA, 1999, pp. 62-68

[Ber96] T. Berners-Lee, R. Fielding, and H. Frystyk, Hypertext Transfer Protocol – HTTP/1.0, IETF, Internet-Draft, Feb. 19, 1996

[Bre98] Eric Brewer, Randy H. Katz, Elan Amir, Hari Balakrishnan, Yatin Chawathe, Armando Fox, Steven D. Gribble, Todd Hodes, Gian Nguyen, Venkata N. Padmanabhan, Mark Stemm, Srinivasan Seshan and Tom Henderson, A Network Architecture for Heterogeneous Mobile Computing, IEEE Personal Communications Magazine 5(5), 1998, pp. 8-24

[Cal98] Kenneth L. Calvert, Samrat Bhattacharjee, Ellen Zegura and James Sterbenz, Directions in Active Networks, IEEE Communications Magazine 38(10), 1998, pp. 72-78

[Cha95] D. Brent Chapman and Elizabeth D. Zwicky, Building Internet Firewalls, O'Reily & Associates, Inc., 1995

[Cha96] M. Chatel, Classical versus Transparent IP Proxies, RFC 1919, Network Working Group, March, 1996

[Cha97] Henry Chang, Carl Tait, Norman Cohen, Moshe Shapiro and Steve Mastrianni, Web Browsing in a Wireless Environment: Disconnected and

Asynchronous Operation in ARTour Web Express, MOBICOM '97, Budapest, Hungary, 1997, pp. 260-269

[Che99] Stephen Cheng, Kevin Lai and Mary Baker, Analysis of HTTP/1.1 Performance on a Wireless Network, Tech Report CSL-TR-99-778, Stanford University, Feb, 1999

[Cro85] Bill Croft and John Gilmore, BOOTSTRAP PROTOCOL (BOOTP), RFC 951, Network Working Group, September, 1985

[Dro93] R. Droms, Dynamic Host Configuration Protocol, RFC 1531, Network Working Group, October, 1993

[Fie97] R. Fielding, J. Gettys, J. Mogul, H. Frystyk and T. Berners-Lee, Hypertext Transfer Protocol – HTTP/1.1, RFC 2068, Network Working Group, January, 1997, http://www.w3.org/Protocols/rfc2068/rfc2068

[For94] George H. Forman and John Zahorjan, The Challenges of Mobile Computing, IEEE Computer 27(6), 1994, pp. 38-47

[Fra97] Larry Francis, Mobile Computing – A Fact in Your Future, SIGDOC 97, Snowbird, Utah, 1997, pp. 63-67

[Gra96] Robert Gray, Davit Kotz, Saurab Nog, Daniela Rus and George Cybenko, Mobile agents for mobile computing, Technical Report PCS-TR96-285, Dept. of Computer Science, Dartmouth College, Hanover, NH, May 2, 1996, ftp://ftp.cs.dartmouth.edu/TR/TR96-285.ps.Z

[Gut99] E. Guttman et al., Service Templates and Service: Schemes, RFC 2609, Network Working Group, June, 1999

[Gut99b] E. Guttman et al., Service Location Protocol, Version 2, RFC 2608, Network Working Group, June, 1999

[Hod99] Todd D. Hodes and Randy H. Katz, Composable ad hoc location-based services for heterogeneous mobile clients, Wireless Networks 5(5), 1999, pp. 411-427

[Hou96] Barron Housel and David Lindquist, WebExpress: A System for Optimizing Web Browsing in a Wireless Environment, MOBICOM '96, Rye NY, USA, 1996, pp. 108-116

[Ibm98] IBM Aglets Software Development Kit – Tahiti User's Guide, IBM Corporation, 1998, http://www.trl.ibm.co.jp/aglets/tahiti1.1/index.html

[Imi96] Tomasz Imielinski, Mobile computing: DataMan project perspective, Mobile Networks and Applications 1(4), 1996, pp. 359-369

[Jos97] Anthony D. Joseph, Joshua A. Tauber and M. Frans Kaashoek, Mobile Computing with the Rover Toolkit, IEEE Transactions on Computers 46(3), March 1997, pp. 337-352

[Jos97b] Anthony D. Joseph and M. Frans Kaashoek, Building reliable mobile-aware applications using the Rover toolkit, Wireless Networks 3(5), 1997, pp. 405-419

[Jos99] Anupam Joshi, On Proxy Agents, Mobility and Web Access, Technical Report 99-02, CSEE Department, UMBC, 1999, http://www.cs.umbc.edu/~ajoshi/dbrowse/mb_ref.html

[Jos99b] Anupam Joshi, On Mobility and Agents, DIMACS Workshop on Mobile Networks and Computing, March, 1999, http://www.cs.umbc.edu/~ajoshi/dbrowse/mb_ref.html

[Kem99] J. Kempf and E. Guttman, An API for Service Location, RFC 2614, Network Working Group, June, 1999

[Kem99b] James Kempf and Pete St. Pierre, Service Location Protocol for Enterprise Networks: Implementing and Developing a Dynamic Service Finder, John Wiley and Sons Inc., 1999

[Kes99] Gary C. Kessler and Carol A. Monaghan, Windows NT and DHCP, Windows NT Magazine, May, 1999, http://www.win2000mag.com/Articles/Print.cfm?Action=Print&ArticleID=5181

[Lan97] Danny B. Lange and Yariv Aridor, Agent Transfer Protocol – ATP/0.1, IBM Tokyo Research Laboratory, Draft 4, March 19, 1997, http://www.trl.ibm.co.jp/aglets

[Lan98] Danny B. Lange and Mitsuru Oshima, Programming and Deploying Java Mobile Agents with Aglets, Addison Wesley Longman Inc., 1998

[Lil95] Mika Liljeberg, Timo Alanko, Markku Kojo, Heimo Laamanen and Kimmo Raatikainen, Optimizing World-Wide Web for Weakly Connected Mobile Workstations: An Indirect Approach, Proceedings of the Second International Workshop on Services in Distributed and Networked Environments (SDNE '95), Whistler, Canada, 1995, pp. 132-139

[Luc98] Lucent Technologies, WaveLAN/PCMCIA Card User's Guide, Lucent Technologies Inc., 1998, http://www.wavelan.com/support/doclib/description.html?id=66

[Mur94] James D. Murray and William VanRyper, Encyclopedia of Graphics File Formats, Oreilly and Associates Inc., 1994

[Nar96] T. Narten, E. Nordmark and W. Simpson, Neighbor Discovery for IP Version 6 (IPv6), RFC 1970, Network Working Group, August, 1996

[Nei98] J. Neilson, Web Predictions for 1999, December. 1998, http://www.zdnet.com/devhead/alertbox/981227.html

[Nel89] Mark Nelson, LZW Data Compression, Dr. Dobb's Journal, October, 1989, http://dogma.net/markn/articles/lzw/lzw.htm

[Nob95] Brian Noble, Morgan Price and M. Satyanarayanan, A Programming Interface for Application-Aware Adaptation in Mobile Computing, Proc. USENIX Symposium on Mobile and Location-Independent Computing, April 1995, pp. 57-66

[Obj97] The Object Management Group, Mobile Agent System Interoperability Facilities Specification, OMG TC Document orbos/97/10-05, The Object Management Group, Framingham, MA, 1997

[Osh97] Mitsuru Oshima and Guenter Karjoth, Aglets Specification (1.0) Draft 0.30, May 20, 1997, http://www.trl.ibm.co.jp/aglets

[Per95] Charles E. Perkins and Kevin Luo, Using DHCP with Computers that Move, Wireless Networks 1(3), March 1995, pp. 341-353

[Per95b] Charles E. Perkins and Tangirala Jagannadh, DHCP for Mobile Networking with TCP/IP, 1'st IEEE Symposium on Computers and Communications, Alexandria, Egypt, July, 1995, pp. 255-261

[Per96] Charles E. Perkins, IP Mobility Support, RFC 2002, Network Working Group, October, 1996

[Per96b] Charles E. Perkins and David B. Johnson, Mobility Support in IPv6, MOBICOM 96, Rye, NY, November, 1996, pp. 27-37

[Per96c] Charles E. Perkins and Harry Harjono, Resource discovery protocol for mobile computing, Mobile Networks and Applications 1(4), 1996, pp. 447-455

[Per98] Charles Perkins, Mobile IP, International Journal of Communications Systems, 11(1), March 1998, pp. 3-20

[Per98b] Charles Perkins, Mobile networking in the Internet, Mobile Networks and Applications, 3(4), 1999, pp. 319-334

[Per99] C. Perkins and E. Guttman, DHCP Options for Service Location Protocol, RFC 2610, Network Working Group, June, 1999

[Pet99] Randal J. Peters, Mobile Computing: Why go Wireless?, 1999, http://www.cs.umanitoba.ca/~randal

[Por98] Thomas F. La Porta, Ramachandran Ramjee, Thomas Woo and Krishan K. Sabnani, Experiences with network-based user agents for mobile applications, Mobile Networks and Applications 3(2), 1998, pp. 123-141

[Sah98] Akhil Sahai and Christine Morin, Mobile Agents for Enabling Mobile User Aware Applications, Autonomous Agents 98, Minneapolis MN, USA, 1998, pp. 205-211

[Sat96] M. Satyanarayanan, Fundamental Challenges in Mobile Computing, Proc. Of the Fifteenth annual ACM Symposium on Principles of Distributed Computing, Philadelphia PA, USA, 1996, pp. 1-7

[Sch96] Bill Schilit, Fred Douglis, David M. Kristol, Paul Krzyzanowski, and John A. Trotter, TeleWeb: Loosely connected access to the World Wide Web, Computer Networks and ISDN Systems 28, 1996, pp. 1431-1444

[Sol98] James D. Solomon, Mobile IP: The Internet Unplugged, Prentice Hall, 1998

[Tan96] Andrew Tanenbaum, Computer Networks 3rd Edition, Prentice Hall, 1996

[Tho96] S. Thomson and T. Narten, IPv6 Stateless Address Autoconfiguration, RFC 1971, Network Working Group, August, 1996

[Tho97] Tommy Thorn, Programming Languages for Mobile Code, ACM Computing Surveys 29(3), September 1997, pp. 213-239

[Voe94] Geoffrey Voelker and Brian Bershad, Mobisaic: An Information System for a Mobile Wireless Computing Environment, Proc. Of the IEEE Workshop on Mobile Computing Systems and Applications, Santa Cruz, CA, 1994, pp. 185-190

[Wav99] WaveLAN/IEEE Turbo 11Mb PC Card Datasheet, Oct. 1999, http://www.wavelan.com/support/doclib/index.html

[Wel97] Girish Welling and B. R. Badrinath, A Framework for Environment Aware Mobile Applications, Proc. IEEE International Conference on Distributed Computing Systems, 1997, pp. 384-391

[Wet98] David J. Wetherall, John V. Guttag and David L. Tennenhouse, ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols, IEEE OPENARCH '98, San Francisco, CA, April 1998, pp. 117-129

[Wim93] W. Wimer, Clarifications and Extensions for the Bootstrap Protocol, RFC 1542, Network Working Group, October, 1993

[Won97] David Wong, Noemi Paciorek, Tom Walsh, Joe DiCelie, Mike Young and Bill Peet., Concordia: An Infrastructure for Collaborating Mobile Agents, Proc. Of the First International Workshop on Mobile Agents, Springer-Verlag, Berlin, 1997, pp. 86-97

[Won99] David Wong, Noemi Paciorek and Dana Moore, Java-based Mobile Agents, Communications of the ACM 42(3), March 1999, pp. 92-102

[Zen99] Bruce Zenel, A general purpose proxy filtering mechanism applied to the mobile environment, Wireless Networks 5(5), 1999, pp. 391-409