WIRELESS COMMUNICATIONS AND MOBILE COMPUTING
Wirel. Commun. Mob. Comput. 2006; **6**:17–34
Published online 21 April 2005 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/wcm.240

A mobile service platform using proxy technology

Ming-Feng Chen,[1] Yi-Bing Lin,[1,*,†] Herman C.-H. Rao[2] and Quincy Wu[1]

[1]*Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu 300, Taiwan*
[2]*Far EasTone Telecommunications, Taipei 114, Taiwan*

Summary

This paper proposes iMobile, a proxy-based platform for developing mobile services for various mobile devices and wireless access technologies. iMobile acts as a message gateway that allows mobile devices to relay messages to each other through various protocols on different access networks. It allows mobile devices to access internet services, corporate databases and to control various network devices. iMobile implements three key abstractions: *dev-let*, *info-let* and *app-let*. An info-let provides abstract view of information space. An app-let implements service or application logic by processing information from various info-lets. A dev-let receives and sends messages through any particular protocols for mobile devices. The *let engine* supports user and device profiles for personalization and transcoding, and invokes proper app-lets and info-lets to answer requests from a dev-let. The iMobile modular architecture allows developers to write device drivers, information access methods and application logic independently from each other. Copyright © 2005 John Wiley & Sons, Ltd.

KEY WORDS: mobile computing; peer-to-peer computing; proxy server; wireless network

1. Introduction

Rapid advances in mobile devices, wireless networking and messaging technologies have provided mobile users a plethora of alternatives to access internet. Examples of these devices and protocols include Palm PDA with web clipping [32], cellular phones with wireless application protocol (WAP) [40], short message service (SMS) [16], e-mail devices (BlackBerry [9], AT&T PocketNet phone [7] etc.) that support POP3 [25] or IMAP [27], Pocket PCs that support AOL instant messenger (AIM) [2] etc. Unfortunately, these approaches do not interwork with each other easily. A mobile user faces the dilemma of desiring the convenience of various mobile accesses to critical services and, in the mean time, suffering from managing the complexity of incompatible devices and user interfaces [35]. Wireless internet is much more complicated than simply accessing the internet wirelessly. Wireless users, being mostly mobile, have different needs, motivations and capabilities from wired users. For example, a mobile user is

*Correspondence to: Yi-Bing Lin, Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu 300, Taiwan.
†E-mail: liny@csie.nctu.edu.tw

Contract/grant sponsor: NSC Program for Promoting Academic Excellence of Universities; contract/grant number: NSC 93-2752-E009-005-PAE

usually in a multi-tasking mode (accessing the internet while attending a meeting or shopping in the mall). On the other hand, a mobile user may not always access the internet wirelessly, and a wireless user may not be mobile at all [39]. The mobile accesses (e.g. checking stock quotes, weather or finding a nearby restaurant) are usually bursty in nature and very task-oriented. To access diverse services, different identities are utilized; for example cellular phone numbers and instant messaging screen names are more meaningful to mobile users than office phone numbers and static IP addresses. In this paper, we propose iMobile [19], a user-friendly environment for mobile internet.

As shown in Figure 1, iMobile runs on a computer with connections to the internet and a wireless modem with two-way SMS. Devices can communicate with iMobile through various protocols and access networks. For example, GSM/TDMA phones with two-way SMS can communicate with iMobile through an SMS driver hosted on iMobile. CDPD devices (such as AT&T PocketNet phone [7] and Palm V with the Omnisky modem [30]) can use WAP to access iMobile through the internet. E-mail devices such as BlackBerry [9] can use the standard e-mail protocols on the CDPD network or a two-way paging network to communicate with iMobile. PC and PDA devices can use AOL instant messenger or web browsers to interact with iMobile. iMobile receives messages and commands from these devices, accesses internet services and information on behalf of the mobile users, and then relays messages or internet content back to the destination devices (which can be different from the sending devices).

Figure 2 illustrates the relationship of iMobile, iProxy and iMobile Micro Edition. The iMobile

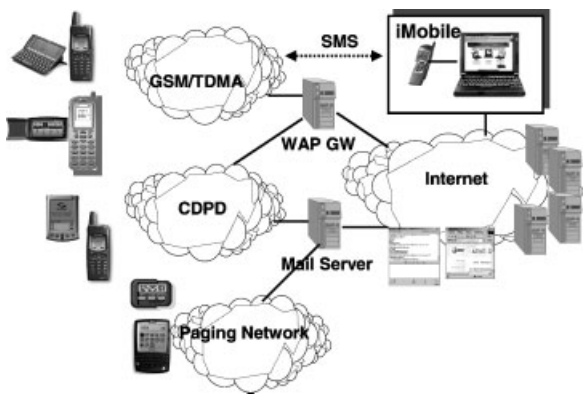


Fig. 1. Personal mobile service network.

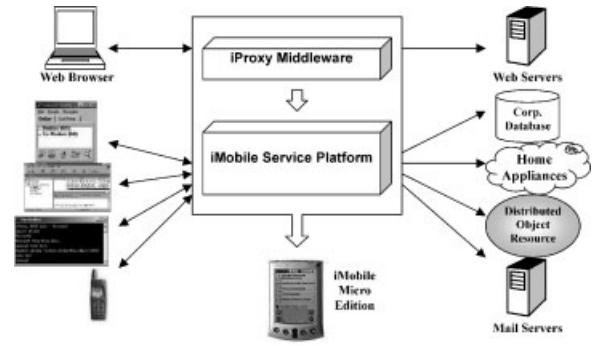


Fig. 2. The relationship of research topics.

architecture hides the complexity of multiple devices and content sources from mobile users. This goal is achieved by utilizing a programmable proxy server called iProxy [8,20]. iProxy provides an environment for hosting agents and personalized services, which are implemented as reusable building blocks in Java. Since iProxy provides a built-in web server, an iProxy agent can be invoked as a regular common gateway interface (CGI) program. It also allows scripts embedded inside web pages, which invoke agents to perform specific tasks. iProxy was originally designed as a middleware between user browsers and web servers. It maintains user profiles and enhances intelligence of a traditional proxy server to provide personalized value-added services [21] such as filtering, tracking and archiving services [22]. iProxy provides customization and personalization features, which are very important for supporting iMobile services.

To support interactions among mobile devices in heterogeneous networks, we propose a lightweight mobile service platform called iMobile Micro Edition (ME), which minimizes the requirement of system resources and is suitable for execution on mobile devices. iMobile ME communicates with each other through a message center called iMobile router, which stores and delivers messages for mobile devices. This architecture provides a platform for iMobile-base peer-to-peer (P2P) computing.

This paper describes the design guidelines and implementation of iMobile. The paper is organized as follows. Section 2 describes related works. Section 3 discusses the iProxy middleware. Section 4 elaborates on the iMobile architecture. Section 5 describes the user and device profiles used in personalization and transcoding services. Section 6 discusses the iMobile-based P2P computing. Section 7 summarizes our work with future directions.

2. Related Works

This section describes related research efforts for personal services and P2P systems.

2.1. Personal Services

The IBM WBI project [33] uses intermediaries to produce and manipulate web content, perform content distillation and implement protocol extensions. WBI includes a transcoding proxy as a web intermediary between web servers and client devices, which adapts varying bandwidths to different client communication links. TranSend [5] is a scalable transformational web proxy, which focuses on efficient data type-specific content distillation. iMobile differs from both approaches by emphasizing how a proxy-based platform can provide a uniform interface (character stream/MIME type) in the dev-let abstraction to deal with a variety of devices and their protocols, which cannot be achieved by a web proxy. The ICEBERG project [17] shares the iMobile goals of any-to-any communication services and personal mobility services, but has so far concentrated mostly on voice, rather than data services.

The Apache Cocoon Project [3] allows automatic generation of HTML, PDF and WML files (for WAP devices) from extensible markup language (XML) files. This feature can be provided in iMobile by integrating the XML and extensible stylesheet language transformations (XSLT) technologies in the transcoding mechanism inside the let engine. Before transcoding, a device profile must be provided to describe the characteristics (size, format etc.) of the receiving device. To address this issue, a W3C working group called CC/PP (composite capabilities/preferences profiles) [28] has created a universal structured format for client device profile that can be accessed by an original server or proxy. We are currently investigating the progress of this protocol and may utilize the CC/PP format for iMobile device profile.

Since iMobile interacts with multiple networks and protocols, it relies on different authentication mechanisms. For device identity, we use the cellular phone IDs on mobile phone network, AOL buddy names on the AIM network and generic user IDs and passwords for WAP, HTPP and Telnet clients. The iMobile platform itself does not have control over how secure these networks are. Solutions such as Charon [4] and the secure shell (SSH) provide end-to-end authentication services. Charon focuses on how to secure the connection between a client and an application-level proxy. This approach allows extremely lightweight and amenable client module to be implemented on PDA and mobile devices. Most of the computations needed to exercise the Kerberos protocol [24] and establish a secure channel are located at the proxy. We are looking into the adoption of a Charon-like implementation for iMobile clients.

Remote control of X10 home network devices is not new. The Aladdin project [44] utilizes e-mail to remotely control the X10 devices. This project concentrates on home automation and the reliability issues. On the other hand, the focus of iMobile is mobility. Home automation is one of the applications built on top of the flexible iMobile platform.

2.2. P2P Communication

iMobile ME P2P model provides a simple infrastructure that involves a network-based application router and a simple device-independent platform on each mobile device. This infrastructure allows communications and resource sharing among mobile devices through message queue synchronization. Our P2P infrastructure resembles Napster [29] in that it relies on a central server, the iMobile router, to reach mobile devices. On the other hand, iMobile differs from Napster in several aspects.

- *Communication mechanisms*: Napster targets on desktop users with internet connections. Through the dev-let abstraction, iMobile ME aims to support multiple communication protocols including HTTP, SMS, instant messaging protocols (AIM or Jabber [23]) etc.
- *Resource access*: Napster focuses on MP3 or WMA music file support. With info-let abstraction, iMobile ME aims to support multiple forms of resources such as location information, camera views and other resources that can be accessed through specialized interfaces in local environments.
- *Queue synchronization*: While most Napster users remain connected during file transfer, the nature of intermittent communication capabilities of mobile devices require a request/response queue synchronization mechanism between the iMobile ME devices and the network-based iMobile router. Requests and responses will not be lost even if the receiving devices are not readily available.
- *Service discovery*: Unlike Napster, the iMobile router does not store an index of info-lets available on all mobile devices. Instead, a primitive help

info-let is provided on each mobile device, which lists the info-lets available on that device. This approach avoids the mountainous cost of collecting all services in a single server.

Gnutella [15] is a P2P system that does not require any central server or database. Each Gnutella peer uses a constrained broadcast mechanism to forward packets to all of its peers with a 'time-to-live' parameter set on each packet. We are adding a group communication mechanism to iMobile ME so that it also supports ad-hoc P2P networking. The group communication module in each iMobile ME instance can identify nearby ME devices and start communicating with each other without relying on any network-based router.

CompanionLink [11] or XTNDConnect Server [13] is mainly designed for data synchronization, with very little resource sharing support among mobile devices. On the other hand, the queue synchronization mechanism provided by iMobile ME allows mobile devices to access enterprise databases or servers through network synchronization.

SyncML [38] is becoming a common language for synchronizing all devices and applications over any networks. SyncML leverages platform-independent XML. This approach is an important step towards standardizing data exchanges among mobile devices. The XML-based data exchange mechanism can be supported in iMobile ME by introducing a SyncML dev-let.

The Pebbles Project at CMU [10] explores how small handheld devices can serve as a useful adjunct to the 'fixed' computers. The CANS Project at NYU [42] aims to provide a user with seamless, ubiquitous access to a service irrespective of the user's end device and location. These projects consider handheld devices as extensions to desktop devices. iMobile ME, on the other hand, provides information access and exchange facilities among mobile devices.

The Pebbles Project focuses on how handheld devices and the PC work together by sharing the (multimedia) user interface. The handheld may take turns controlling the PC's cursor and keyboard input or show the thumbnail of the PC's screen. In Pebbles environment, handheld devices are considered as the external controllers of the desktop PC. On the other hand, iMobile ME is a lightweight platform installed on a handheld device, which exports local information stored only in the device. The server application runs on the handheld device rather than on a desktop PC. In iMobile ME, each device will process a request issued by another device and send back the result data. We focus on providing an appropriate platform for building services on handheld devices.

The Berkeley Ninja Project [36] developed an interesting approach for secure service discovery based on a PKI infrastructure and a capability manager. Ninja does not require a central server to verify a user's access rights during service invocations. For iMobile ME devices with support for the Java cryptography architecture (JCA), such as Java crypto and security implementation (JCSI) Micro Edition [41] secure sockets layer (SSL for the J2ME connected device configuration (CDC) and Personal Java), security infrastructure similar to Ninja can be implemented.

3. iProxy Middleware

iProxy [8,20] is a middleware for web applications, which can be installed as a personal proxy server running on an end-user's machine. iProxy provides standard web proxy functions for accessing, caching and processing web data. It allows users to select the routes to access web servers (e.g. choosing different proxies for different hosts), archive web pages, record and analyze web access history, and manage the proxy's cache. iProxy also provides hooks to plug in new functions for processing data received from web servers. For example, the web data can be condensed, compressed, encrypted or patched. The iProxy filters convert pages back to their original forms, or synthesize new pages from web data and personal information stored on the local disk. With a built-in web server, iProxy can accept HTTP calls from internet and trigger local applications to perform tasks such as event notifications, alerts and data pushing.

3.1. iProxy System Architecture

As shown in Figure 3, iProxy consists of four components: proxy, web, walking/scheduling, and connection management. The proxy component receives a uniform resource locator (URL) request from web browser, forwards the request to the corresponding web server, stores the returned page in the cache and forwards it to the browser. The web component allows user to access the iProxy configuration, invoke CGI programs or execute a script embedded in a web page. The walking/scheduling component provides functions to trace the HTML tree structure asynchronously (e.g. through background tracing or periodic tracing).

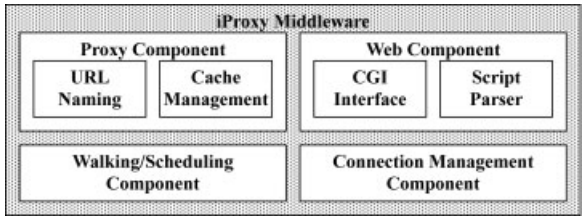


Fig. 3. iProxy architecture.

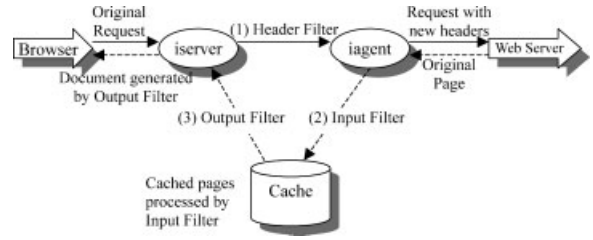


Fig. 4. Proxy component filters.

The connection management component handles the socket connections for web accessing, inter-iProxy communication and TCP/IP socket forwarding.

3.1.1. Proxy component

iProxy maintains a cache to store frequently accessed web pages. When the proxy component receives a URL request from user browser, it returns the cached page for a cache hit. For a cache-miss access, the proxy component forwards the request to the corresponding remote web server. When requested page is returned, the proxy component stores the page in the cache, and then forwards it to the user. The proxy component consists of *URL* naming and cache management sub-components. To provide powerful add-on services, the URL naming sub-component extends the protocol exercised between the browser and web server. Two new specifications, *action* and *view* parts, are added to the URL format, which result in the following new format: *http://*view@*hostname/filepath/filename.html?iProxy&action* = ...

The action specification, starting with '?iProxy', notifies iProxy to perform specific actions on the web page. An action may archive a page with timestamp, store the page in a specific package or invoke a CGI program to modify the page. The view specification is used to access web pages cached at a particular time or in a particular package. Examples of action/view will be given in Subsection 3.2.1 and 3.3.

The cache management sub-component maintains the cache repositories for web pages obtained from remote web servers. It may only keep the newest pages, as a normal proxy server does, or keep multiple versions according to the timestamps. Figure 4 shows the caching processes. For a web request, the *iserver* process receives the request and creates an *iagent* process to acquire and cache the web page. After the web page is cached, the iserver process forwards the cached page to the browser. The cache management sub-component may invoke three different filters in the caching processes: *Header*, *Input* and *Output Filters*.

1. The *Header Filter* modifies the URL requests received from the web browser. This filter may add new cookies, header fields or modify the URL to change the accessing behavior. For example, changing the URL of a web image to a local image eliminates the transmission overhead due to remote graphical accessing. This filter is useful for mobile devices with low bandwidths.
2. The *Input Filter* modifies a page before storing it into the cache. This filter provides the flexibility to customize, compress, encrypt or translate a cached page.
3. The *Output Filter* processes a cached web page before it is returned to the browser. This filter may attach additional HTML components into the cached pages, including system statistics or personal information. The output filter is also used to decompress, decrypt or translate a cached page.

3.1.2. Web component

The web component consists of two sub-components: *CGI interface* and *script parser*. A URL request to iProxy is forwarded to the web component. If the request is a CGI program, the *CGI interface* sub-component invokes the corresponding Java program. On the other hand, if the request is a script file starting with '*#!/iProxy/script*' (see the example in Figure 17), the script parser sub-component interprets the script and generates the page from the output of the script. The web component also exports web interface to a user for accessing iProxy system resources indicated by a URL starting with '*http://localhost/*'. This root page (http://localhost/) is the entry point for accessing the system parameters and configurations.

Examples of interactions between the proxy and web components are shown in the filter programs describe in Subsection 3.1.1. The proxy component

defines rules for filter programs to indicate when and which filter should be invoked on a specific URL request. These rules reference the filters as CGI programs defined in the web component.

3.1.3. Walking/scheduling component

The walking/scheduling component traces the HTML tree structure and stores the pages in the cache or archive repository. An HTML walking is invoked by the URL action specification or through the administration web pages (i.e. http://localhost/). Starting from a root page, the walking component parses the HTML structure, and tracks the hyperlinks and/or images in the page. The walking parameters determine the depth of tracing, the images for caching and the pages to be traced locally (i.e. within the same web site) or globally.

A walking result may be stored in three different repositories: cache, archive or package. The cache repository keeps the newest cached pages, the archive repository maintains multiple versions with time-stamps and the package repository stores all walked pages in a single file. Caching the walking pages speeds up subsequent accesses. This feature also effectively supports off-line browsing because it keeps not only the visited pages but also the subsequent hyperlinks in the cache. The archive repository memorizes the historical web pages for tracking, searching and comparison. The package repository maintains the pages in different packages. A package can be moved around various servers to support user mobility. These repositories are handled by the cache management component and can be accessed using the view specification described in Section 3.1.1.

3.1.4. Connection management component

The connection management component handles the socket connections for web accessing, inter-iProxy communication, and TCP/IP socket forwarding. Figure 5 shows examples of routing paths for web access. In this example, an iProxy server *iProxy1* and a corporation web server *Corp. Web* are located in the same LAN. In the internet, there are another iProxy server *iProxy2*, a standard proxy server *Proxy B*, and several web servers *A1*, *A2*, *B1*, *B2* and *C*. *iProxy1* specifies *routing rules* to handle the requests for different web servers. For example, the routing rules can be:

Rule 1: Forward the requests for *A1* and *A2* to *iProxy2*.
Rule 2: Forward the requests for *B1* and *B2* to *Proxy B*.

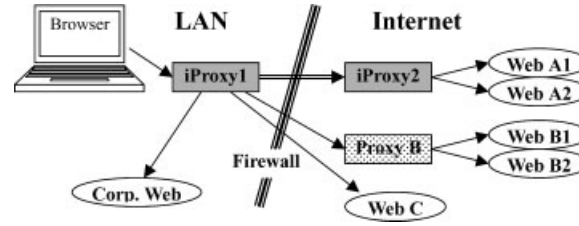


Fig. 5. Web access routing and inter-iProxy communications.

Based on the above rules, *iProxy2* handles the forwarded requests for *A1* and *A2*, Proxy B handles the forwarded requests for *B1* and *B2*, and *iProxy1* handles the requests for *Corp. Web* and *Web C*.

These routing rules are specified in the administration pages, which provide the flexibility to forward requests for particular web servers through other proxy servers. They are especially useful when the connections between proxy servers support the *keep-alive* feature. In Figure 5, *iProxy2*, *A1* and *A2* are located in the same network. These servers are remote from *iProxy1*. The connection setup overhead will be large if *iProxy1* creates new connection to *A1* and *A2* for every request. To reduce the overhead, *iProxy1* reuses the inter-iProxy connections between *iProxy1* and *iProxy2* and forwards multiple requests and replied pages in one connection.

3.2. Personal Services

Novel services can be provided by a client-side iProxy that has access to a user's private information; for example the web access history and personal finance information. This section describes a few personal services that we have implemented.

3.2.1. Personal web archive

Although existing search engines allow users to find current pages, they may not allow locating and viewing the pages that were accessed in the past, except for those that are still kept in the browser cache. Thanks to inexpensive mass storage, a client-side iProxy can afford to archive all web pages that have been viewed before. These pages can be retrieved later without even bookmarking them. Tools like AltaVista Discovery [1] can be used to index and search these web pages. Our experience showed that an active web browser user could create a web archive of roughly 80–100 MB per month, including images and all downloaded documents. This amounts to about 1 GB

of storage per year for all pages a user has seen, which is now affordable to many customers.

Because iProxy intercepts HTTP requests, it can effectively extend a URL name space by adding a timestamp in front of the regular HTTP address. For example, the URL of the 2003 February 11th version of AT&T's website is *http://+20030211@ www.att.com/*. Even if the AT&T website goes through major redesigns, the *persistent* URL (indexed by the timestamps) will always provide the same content.

3.2.2. Personal web page reminders and hot sites

A client-side iProxy can analyze the logged web pages to provide convenient browsing services. Figure 6 shows two services that we implemented.

- *TO-READ homepages*: A user specifies the list of websites and corresponding frequencies they should be visited. iProxy checks the last visited dates and schedules a list of pages that will be visited by the user today.
- *HOT sites*: iProxy computes the number of visits to each website and lists the top ten websites with their last visiting dates whenever the user accesses the personal portal. The hot-site list allows a user to retrieve the last visit time of a favorite site (the timestamp is displayed along the website link), access the latest version by clicking on the link, and compare the new version with the old one by using tools such as WebCiao [45] or AIDE [14].

3.2.3. My stock portfolio

Most portals allow users to specify the interesting stocks and display the latest prices when the user accesses the personalized page. However, these portals cannot compute personal current balance or net gain/loss unless the user provides confidential information such as the number of owned shares and the purchased dates. Such practice is certainly not convenient to many users. Figure 7 shows a typical portfolio view on Your WorldNet [6]. In this example, the user provides the following fake information in the portal server (see the upper table in Figure 7): one share for each of the AT&T, E*Trade and Netscape stocks. No commission fees were paid and each stock was bought at $30.00.

Suppose that the real purchase price, commission fees and the share number of each stock are stored on the client machine. By constructing an output filter for the stock page, iProxy can retrieve the private information and combine it with stock quotes provided by the remote portal site to compute the balance and net gain/loss. The output filter instructs iProxy to apply the Java class portfolio on the local server whenever the browser issues the corresponding HTTP request. The real numbers are visible to the client only (see the

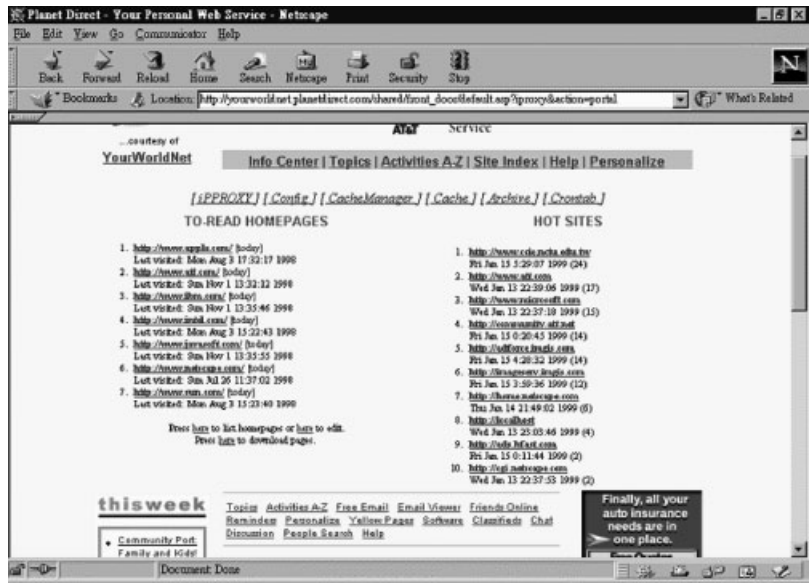
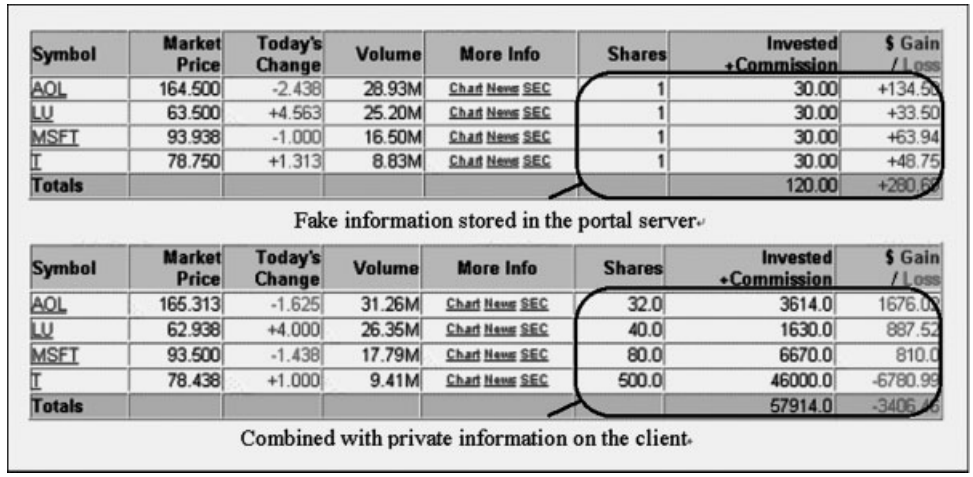


Fig. 6. TO-READ pages and HOT SITES.



Symbol	Market Price	Today's Change	Volume	More Info	Shares	Invested +Commission	$ Gain / Loss
AOL	164.500	-2.438	28.93M	Chart News SEC	1	30.00	+134.50
LU	63.500	+4.563	25.20M	Chart News SEC	1	30.00	+33.50
MSFT	93.938	-1.000	16.50M	Chart News SEC	1	30.00	+63.94
T	78.750	+1.313	8.83M	Chart News SEC	1	30.00	+48.75
Totals						120.00	+280.69

Fake information stored in the portal server

Symbol	Market Price	Today's Change	Volume	More Info	Shares	Invested +Commission	$ Gain / Loss
AOL	165.313	-1.625	31.26M	Chart News SEC	32.0	3614.0	1676.00
LU	62.938	+4.000	26.35M	Chart News SEC	40.0	1630.0	887.52
MSFT	93.500	-1.438	17.79M	Chart News SEC	80.0	6670.0	810.0
T	78.438	+1.000	9.41M	Chart News SEC	500.0	46000.0	-6780.99
Totals						57914.0	-3406.48

Combined with private information on the client

Fig. 7. My portfolio with/without private information.

lower table in Figure 7), which are not revealed to the external portal server.

3.3. Portal Script: Putting the Pieces Together

A proxy-based portal integrates the contents stored in a proxy server with those provided by a regular portal server such as Your WorldNet. Consider the following scenario where a portal server works in concert with a client-side iProxy. A user sends an URL with a proxy directive, *http://www.att.net/?iProxy&action = portal*, through the browser. iProxy first retrieves the homepage from *www.att.net*. This homepage has encoded iProxy directives that are used to process the local data and merge them with server content. iProxy then presents the personal portal page to the user. In order to provide a non-intrusive environment to other users who are not using iProxy, we embed the iProxy directives in HTML comments to generate the portal page. Consider the directives listed in Figure 8.

iProxy intercepts these directives and performs necessary actions before returning the portal page. The directive *version* prints out the version number of iProxy. The directive *to-read* constructs the list of web pages scheduled to be read. The directive *dolog* analyzes the current web access log to produce the statistics. Finally, the directive *top10* presents the results on the personal portal. Browsers without iProxy support will ignore all directives embedded in the HTML comment.

```
<!------- with iproxy only --------->
<!---      :portal version
           :portal to-read
           :portal dolog
           :portal top10          -->
```

Fig. 8. A portal script example.

4. iMobile Service Platforms

Based on the iProxy middleware, iMobile provides a platform to support personalized mobile services. This platform aims to hide the complexity of multiple devices and content sources from mobile users. As shown in Figure 9, iMobile adds three agent abstractions to iProxy: *dev-let*, *info-let* and *app-let*. These components communicate with each other through the *let engine*. This section describes the detailed interactions among these let agents in the iProxy environment.

4.1. Dev-Let

A dev-let is a device controller, which provides various protocol interfaces to user devices. Each dev-let interacts with the let engine through a well-defined interface. It receives user requests (character streams), and returns results in a Multipurpose Internet Mail Extensions (MIME) type acceptable by the receiving device. Figure 9 shows four dev-let examples: AIM, SMS, SMTP/POP3/IMAP and TCP/IP. The AIM dev-let is an AIM client, which receives personal messages as requests and returns the result messages to the sender. The SMS dev-let uses SMS in GSM networks

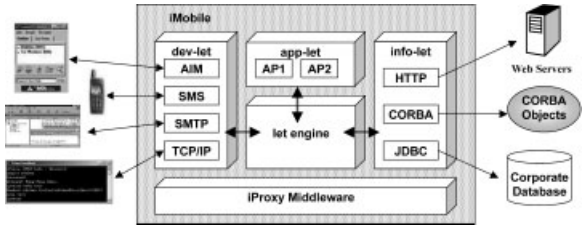


Fig. 9. The iMobile architecture.

for message delivery. The mail dev-let is a mailer that receives requests from mail server using the Post Office Protocol 3 (POP3) and/or Internet Message Access Protocol (IMAP), and sends results to the e-mail address of the sender using the Simple Mail Transfer Protocol (SMTP). The TCP/IP dev-let accepts socket connections from internet, which interacts with mobile users as a character console. The let engine manages these dev-lets and communicates with various mobile devices through these dev-lets.

When iMobile is started, a dev-let for each communication protocol is created, which listens to incoming requests delivered through that particular protocol. For example, the AIM dev-let starts an AIM client and listens to instant messages sent from other AIM clients.

The device driver for a particular protocol may colocate with the dev-let or it may communicate with the dev-let through a TCP-based protocol. This approach allows a device driver to run on a remote machine other than the iMobile server. Figure 10 shows an example where an SMS dev-let communicates with the GSM cellular phone attached to a remote PC through an SMS driver [18]. The protocol for the

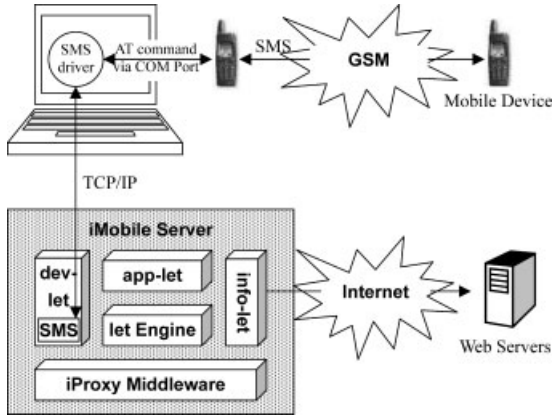


Fig. 10. iMobile communication path for an short message service (SMS) device.

SMS driver is AT Command [12]. Mobile users send short messages to this cellular phone. The cellular phone then forwards the messages to iMobile for processing.

Current iMobile version supports dev-lets that understand protocols including SMS, IMAP, AIM, ICQ and Telnet. iMobile also supports WAP and HTTP through the iProxy HTTP interface. To allow email access to iMobile, the e-mail dev-let periodically monitors messages arriving at a particular e-mail account. A Telnet user can enter iMobile commands through a typical Unix or Windows terminal.

While all iMobile devices and the supported protocols have different user interfaces, every dev-let interacts with the let engine in a standard way. Naming of each device or destination address follows the URL naming format; i.e. protocol name followed by an account name or address. Examples for destination addresses include sms: + 19737086242 (GSM phone), aim: sunshine4 (AIM buddy name), mail: iProxy@research.att.com (email id) etc. Suppose that an iMobile user queries the AT&T (T) stock price. The user invokes the 'quote' app-let by issuing the following message to iMobile:

quote T

If the request is sent using SMS on the GSM network, then the result will be returned as plain text to the receiving GSM phone. On the other hand, if the mobile user wants to forward the result to the email account herman@research.att.com, then the GSM phone issues the following command:

forward mail:herman@research.att.com quote T

Since that e-mail account understands the MIME type TEXT/HTML (according to the device profile to be elaborated later), the result will be delivered as an HTML file (which may include graphics) to herman@research.att.com.

The dev-let abstraction allows users in different networks to easily communicate with each other. For example, if a GSM subscriber wants to send a message to an AT&T PocketNet mail account chen@mobile.att.net on the CDPD network, and also an AT&T TDMA phone 908-500-6531 (Chen's cellular phone) using SMS, then the sender can use the message relay service supported by the 'echo' app-let:

forward mail:chen@mobile.att.net,sms: + 1908500 6521 echo call your boss

In this example, the sender really wants to reach a person, not a device. Since iMobile can map the user

to devices (see Subsection 5.1), and it keeps track of the user's last access from a particular communication channel, we can use an alias to reach either all devices or the last device being used by Chen.

4.2. Info-Let

The info-let abstraction extends the HTTP protocol and URL name space to provide abstract views of various information spaces. An info-let may retrieve or modify an information space. Retrieved information may be passed to an app-let for further processing. Examples of information spaces are given below:

- *Stock quote, weather, flight schedule etc.* are commonly available on many websites, but it would be better to retrieve such information from XML files or databases directly. Figure 11 shows an AIM client Chen that talks to an iMobile AIM agent. Chen issues the 'flight 001' command to query the flight information on the NorthWest airline. The output includes time and gate information for each leg of the flight. The mapping from the flight command to the NorthWest airline is controlled by an app-let that consults the user profile of Chen. Also, the let engine invokes necessary transcoding to tailor the elaborate content on the website to a format appropriate for the receiving AIM device. Figure 12 shows a Palm V (with a wireless modem) that just sent an email to the iMobile email dev-let (imobile@research.att.com) with the command 'sitenews att'. This command instructs iMobile to access the service provided by AT&T's Website News, which reports today's new hyperlinks on AT&T's website (http://www.att.com). The result is sent back as an email formatted for the Palm V PDA.

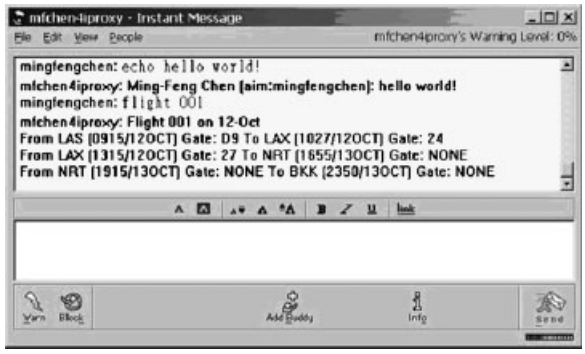


Fig. 11. Flight schedule service.

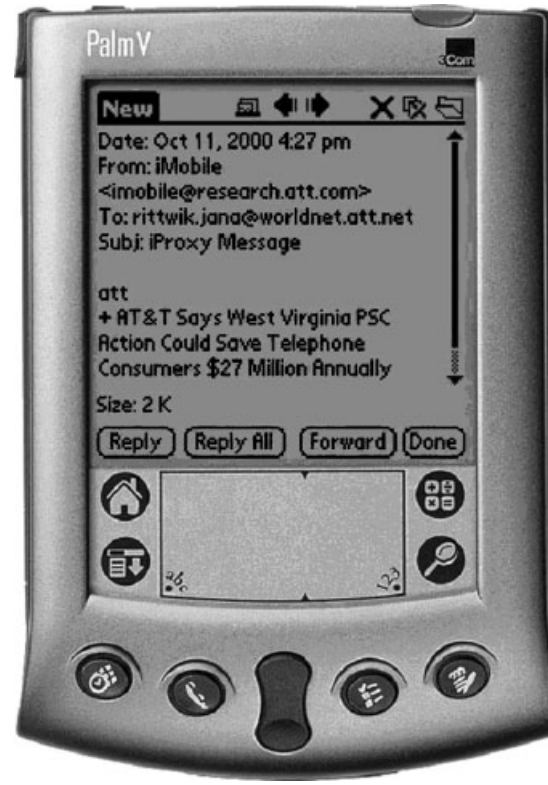


Fig. 12. News service on Palm V.

- *Corporate database* is typically accessed through the Java Database Connectivity (JDBC) and Open Database Connectivity (ODBC) interfaces. iMobile hosts a JDBC info-let that allows mobile users to access or update enterprise database information (employee data, marketing/sales data, system interface data etc.) through SQL-like queries. For example, Figure 13 shows how a user accesses an enterprise database through an AIM client to find

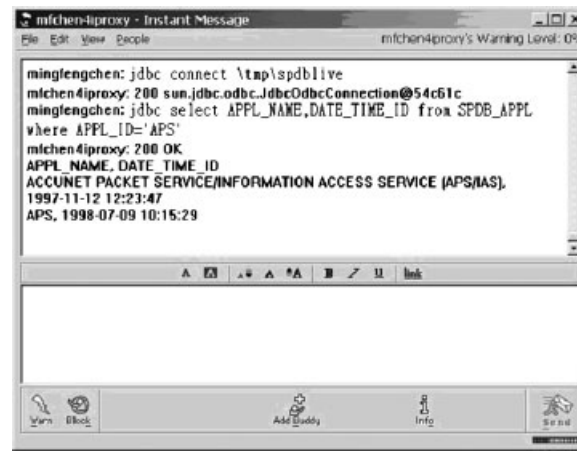


Fig. 13. Corporate database access.

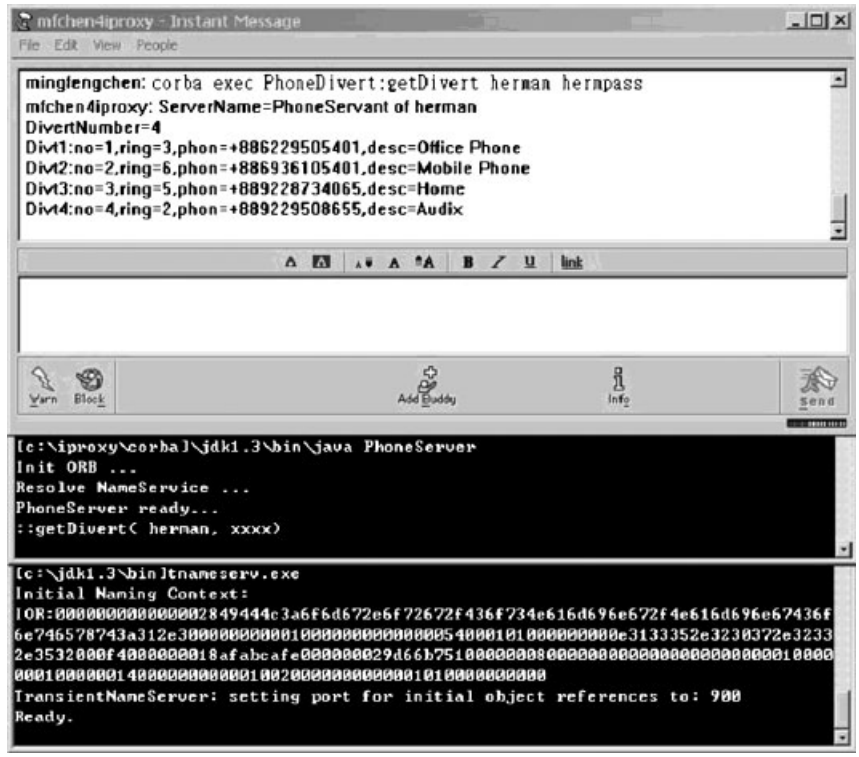


Fig. 14. Common object request broker architecture (CORBA) object access.

the work telephone number of a particular service. One can also access the same information from a PDA that supports AIM. In any case, it is critical that only end-to-end solutions are used for mobile access to corporate databases.

- *Network/infrastructure resources* are typically accessed through the Common Object Request Broker Architecture (CORBA) [31] interface. CORBA is an Architecture and specification for managing distributed program objects in a network. It allows programs at different locations to communicate through an 'interface broker'. iMobile hosts a CORBA info-let that allows mobile users to request services from CORBA objects. Figure 14 shows how an AIM user obtains phone diversion information for the user Herman through the CORBA info-let that accesses a phone server.
- *X10 home network devices* for home appliances (such as lamps and the garage door opener) are connected on the same power line. The X10 [34] device control signals are issued by a computer, and are delivered through the power line. iMobile hosts an X10 info-let that determines when and how to activate certain X10 devices for home environment control. Figure 15 shows how an iMobile user

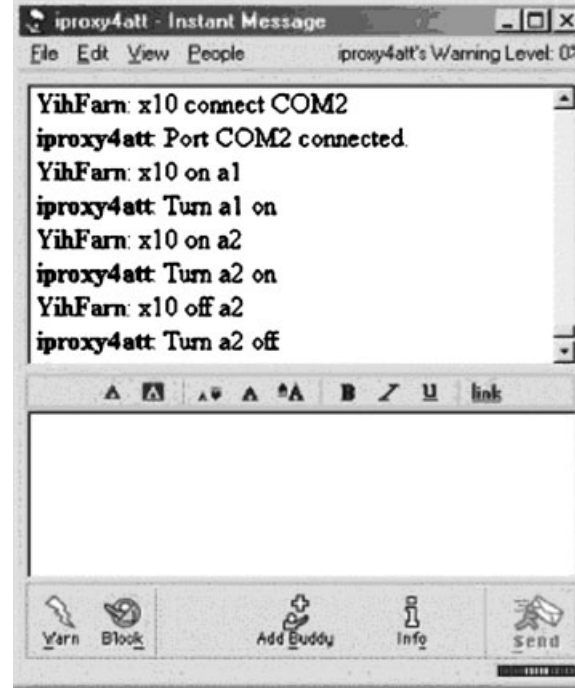


Fig. 15. X10 home network access.

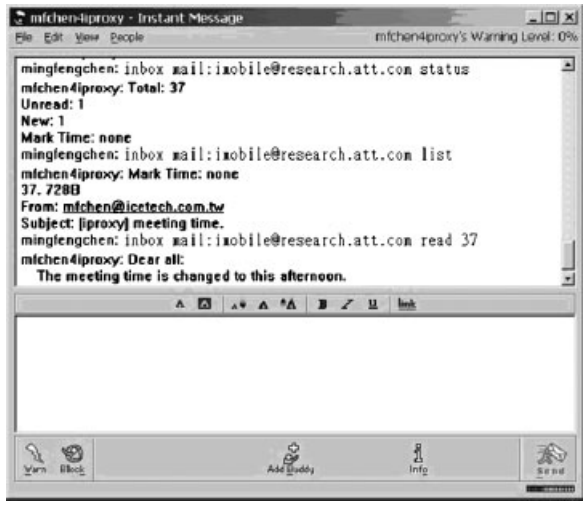


Fig. 16. E-mail service.

controls X10 devices remotely. The user instructs iMobile to locate firecraker (a device that is capable of sending a radio signal to a transceiver device on the X10 network) through the serial port COM2 on the iMobile server. After the connection is established, the user sends the command 'x10 on a1' to turn on the fan (which is named device a1 on that particular X10 network) and 'x10 on a2' to turn on the coffee pot. The X10 interface on iMobile allows a mobile user to remotely control the home appliances from anywhere in the world with a cellular phone, an instant messaging client, or any mobile device supported by iMobile. This example demonstrates that an info-let can be used to both retrieve and change the state of an information space.

- *Mail servers* are managed by an IMAP info-let called inbox that can access a user's e-mail account. In this scenario, encrypted email password is required for user authentication. In Figure 16, a mobile user checks the unread messages in his inbox. He/she then looks at the size (e.g. message 37 has 728 bytes), subject and the sender of every message before actually viewing it. Such interaction style is typical for a mobile user using a communication device with limited bandwidth and screen space.

4.3. App-Let

An app-let implements service or application logic that processes contents from different sources and relays the results to various destination devices through dev-lets. An app-let may have complex interactions with info-lets. Figure 17 shows a FindMeAMovie app-let implemented as an iProxy script. The app-let finds local theaters showing the top ten movies by executing the following steps:

```
#!/iproxy/script
# get the localtion information (zip)
:javabin infoLet zip getlocation
# get top10 movies (mlist)
:javabin infoLet mlist top10 movie
:foreach mtitle ${mlist}
# List the movie title
-- Movie: ${mtitle} --
:javabin infoLet thlist findTheater ${mtitle} ${zip}
:foreach theater ${thlist}
# List the theater
${theater}
:endfor
:endfor
```

Fig. 17. Find-Me-A-Movie app-let.

- obtain the location (zip code) of the cellular phone through the mobile location service,
- identify the top ten movies from a movie database or website,
- for each of these movies, check if any local theater shows that movie and
- list the theaters.

5. User and Device Management

When the let engine receives commands from user devices, it translates the commands according to user aliases and profiles. This section describes device and user profiles in detail.

5.1. Device Profile and Device-to-User Mapping

Every abstract device must register its profile information with the let engine. The format of a device name is *protocol:acct_id*. For example, an AIM device for a user webciao is *aim:webciao*. A device profile is a list of (case-insensitive) attribute-value pairs. The most important attribute is dev.format.accept, which determines the MIME type to be accepted by the device. iMobile uses this information to transcode original content to an appropriate format for this device. iMobile maintains a default profile for each device type. A device instance can overwrite the default profile with device-specific information. Consider the default profile for an email device. The profile has the following content:

```
dev.format.accept=text/html,*/*
dev.page.size=0
```

This profile indicates that the default MIME type is TEXT/HTML, but all MIME types (*/*) are acceptable. Also, the page size '0' means that there is no size limit for each message transmission. These values are inherited by all mail devices unless they are overwritten. For example, the above default values are not appropriate for e-mails used in cellular phones (say, AT&T's PocketNet phone). The device profile for that cellular phone uses the following rules:

```
dev.format.accept=text/plain
dev.page.size=230
```

which indicate that only the MIME type text/plain is appropriate and the phone does not accept messages longer than 230 characters. The device profile may also specify how and when to access this device. For example, a profile may include the following entries:

```
mail.store.checktime=10000
mail.store.url=imap://imobile:password@bigmail
mail.transport.url=smtp://bigmail
```

which specifies the frequency (every 10 s) for checking the e-mail account (store.checktime), the account password (store.url), the transport protocol for sending email (transport.url) etc.

Each device is mapped to a registered iMobile user. There are two reasons for this mapping:

- to ensure access for legitimate iMobile users and
- to personalize a service based on the user profile.

Examples of device-to-user mappings are shown below:

```
sms:+886936731826=herman
sms:+19087376842=chen
mail:dchang@research.att.com=difa
aim:webciao=chen
```

5.2. User Profile

iMobile authenticates a mobile user depending on the device or protocol used by that user. An example of the iMobile user profile is given in Figure 18.

In this example, the user profile stores the user name, password and a list of the devices that the user registers with iMobile. It also stores command and address aliases. When a user accesses iMobile through AIM using the ID webciao, iMobile determines from the user-device mapping that the user is Chen. Therefore, iMobile will use the user profile of Chen to handle all later service requests from this device. For

```
name=Chen
password=xf2gbH3
default=$mail.1
# my addresses
sms.1=sms:+19087376842
mail.1=mail:chen@research.att.com
mail.2=mail:imobile@mobile.att.net
mail.all=$mail.1,$mail.2
aim.1=aim:webciao
# command aliases
sms.cmd.q=quote
sms.cmd.sn=sitenews
# address aliases
sms.addr.cc=aim:chrischen
```

Fig. 18. A user profile example.

example, a user may send the following short message using a GSM phone:

```
forward $mail.1 Q T
```

In this short message, the special character '$' requests iMobile to map the named device (mail.1) to the corresponding entry in the profile. According to the user profile in Figure 18, iMobile interprets the short message as

```
forward mail:chen@research.att.com quote T
```

This user profile also stores the user's last access device in the default parameter. Other mobile users may send the following message to reach the user:

```
forward $chen echo call your boss
```

The alias ('$' + username) requests iMobile to lookup the last access device (mail.1) of Chen and interpret the message as

```
forward mail:chen@research.att.com echo call your
boss
```

As a final remark, iMobile assumes that wireless networks (such as Voicestream GSM or AT&T TDMA networks in North America) are reliable, which provide legal cellular phone IDs through short messages. This assumption is generally acceptable unless a cellular phone is stolen and the user did not lock the phone with a secured password. iMobile also trusts the AOL authentication for non-critical services. Extra user authentication through iMobile is required if the user accesses iMobile through Telnet,

WAP or HTTP. The authentication information should be stored in the user profiles.

6. iMobile-Based P2P Mobile Computing

Many resources available to a mobile device may not be readily available on any networked servers. Examples of the resources include location information, locally captured media files and its exposure to surrounding resources, such as thermometers or X10 cameras that are wirelessly connected. With these resources, the P2P computing paradigm [26] will also enable mobile devices to directly exchange information with each other.

Figure 19 shows an example of P2P mobile computing environment among four mobile users located in different locations with various access networks and devices. The mobile user Chen is in Paris with an iPAQ connected to the internet through wireless LAN, possibly provided by a coffee shop. He may want to access a specific image captured by his friend Wei in New York with a Palm device connected to CDPD/TDMA network. Chen is also interested in the location information of another user in San Antonio, and the address information of a Paris friend, stored in the mobile device owned by his friend in San Diego. All these contents, stored in individual mobile devices, are not available on any network-based servers. To access the contents, Chen must send the requests to other mobile devices. Because these mobile devices may not always connect to the internet, we introduce iMobile router, a network based server that locates the mobile devices and routes the messages among them.

The iMobile-based P2P computing proposes a lightweight service platform called iMobile ME, which provides personalized services on the mobile devices. iMobile ME is a simplified version of the iMobile platform described in the previous sections (which is referred as the standard iMobile in this section). iMobile ME minimizes the requirement of system resources and is suitable for execution on mobile devices.

As shown in Figure 20, iMobile ME consists of two agent abstractions: dev-let and info-let. These components communicate with each other through the let engine. The let engine, dev-let and info-let perform the same functions as those in the standard iMobile platform. The major differences between the iMobile ME and standard iMobile platform are described below.

1. *Data encoding*: To support flexible output format in standard iMobile, the info-lets need to generate the results in MIME format and provide a transcoding mechanism if the result type is not acceptable to the destination device. In iMobile ME, the dev-lets and info-lets only support plain text. This simplification reduces the communication bandwidth and the effort of data processing on mobile devices.
2. *Removal of app-let*: In standard iMobile, an app-let implements application logic by processing information obtained from one or more info-lets. This powerful abstraction allows creation of complicated services by using the iProxy scripts that invoke functions defined in info-lets. Therefore, the info-lets must provide many functions to support the app-lets. On the other hand, to reduce the overhead of processing the requests, iMobile ME removes the app-let component and the script parser. This simplification allows more straightforward execution.
3. *Remote access*: iMobile ME introduces a remote agent (RA) dev-let and a remote procedure call (RPC) info-let to support remote access among

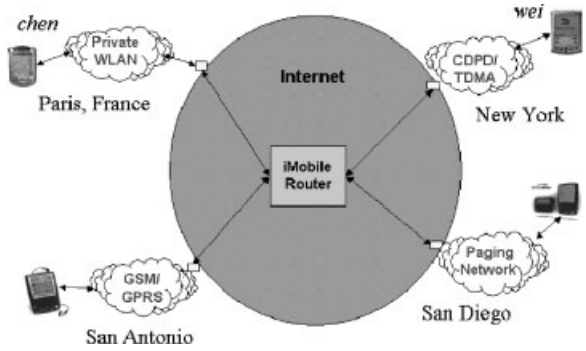


Fig. 19. iMobile-based peer-to-peer (P2P) mobile computing.

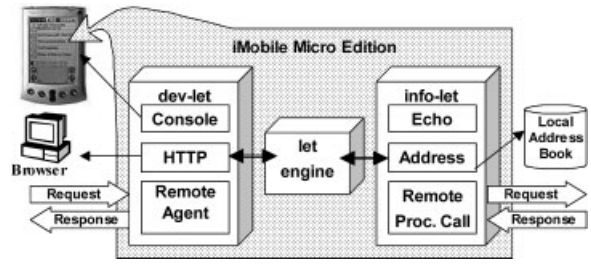


Fig. 20. iMobile micro edition (ME) system architecture.

mobile devices. The RA dev-let accepts the requests from other mobile devices and returns the results to the senders. The RPC info-let forwards the local requests to the remote mobile devices and returns the results obtained from these remote devices. RA and RPC are not found in standard iMobile.

4. *Message queuing*: A mobile device may be disconnected or connected with limited bandwidth, and it may be difficult to retain a long communication session between two interacting mobile devices. Therefore, each remotely accessible dev-let/info-let is extended with an inbox queue which accumulates incoming messages and an outbox queue which buffers outgoing messages.
5. *Message routing*: iMobile ME communicates with the iMobile router to exchange queued messages. The iMobile router is a network-based server, which routes requests and responses among mobile devices. It stores the messages in the queues for every iMobile ME and synchronizes with the queues of all iMobile MEs.

Based on iMobile ME, examples of P2P services and the details of queue synchronization are elaborated in this section.

6.1. iMobile ME Services

Figure 20 shows examples for iMobile ME dev-lets and info-lets implemented in Java 2 Micro Edition (J2ME) [37]. An iMobile ME provides two basic dev-lets: console and RA. The console dev-let provides a pure-text console to send requests and display responses. The RA dev-let receives requests from other mobile devices and returns the results to these devices. If the mobile device is powerful enough to run a simple web server (e.g. iPAQ), iMobile ME may also provide a HTTP dev-let to receive requests directly from web browsers.

An info-let exports the local resource of a mobile device to other devices. Some examples are listed below.

- The Echo info-let simply echoes the received string. This info-let is useful for checking the system and connections among mobile devices. The Echo info-let also provides round-trip delay statistics from one mobile device to another.
- The Address info-let provides a lookup interface for the address book database, which can be found in most mobile devices. Figure 21 illustrates the response shown on a Palm device for address lookup of a user Huang.
- The RPC info-let parses a request to obtain the destination and command parameters. Based on the parameters, the RPC forwards the command to the corresponding destination.
- The Sensor info-let exposes the location or environment information of a mobile device that has a built-in location determination system or a sensor to obtain its surrounding environment information (such as temperature and moisture).

Fig. 21. An address info-let example.

6.2. Queue Synchronization

iMobile ME stores the incoming and outgoing messages in queues and synchronizes with the queues of other iMobile MEs defined in the iMobile router. Queue synchronization is issued by an iMobile ME when it connects to the internet. Every iMobile ME registers a unique ID to the iMobile Router and uses this ID to communicate with each other. Figure 22 shows an example of remote procedure call from iMobile ME1 to ME2, which includes four synchronization actions

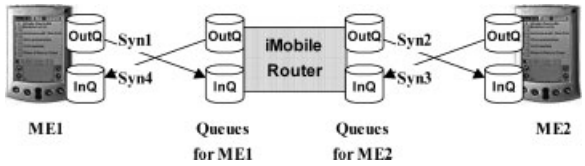


Fig. 22. Queue synchronization example.

1. A RPC request is issued from the console dev-let in ME1. The RPC info-let receives the request and stores it in the outbox queue. Then ME1 issues the first synchronization that forwards the request to the iMobile router. The iMobile router buffers the request in the outbox queue for ME2 and waits for ME2 to retrieve the request.
2. When ME2 issues the second synchronization, it obtains the request from the iMobile router. ME2 executes the request by invoking the corresponding info-let, and stores the response in its outbox queue.
3. ME2 issues the third synchronization that sends the response to the iMobile router. The iMobile router stores the response in the outbox queue for ME1.
4. ME1 issues the fourth synchronization to receive the response. Finally, the console dev-let shows the response on the screen of ME1.

Figure 23 shows a RPC request example issued by Chen to look up address book for Huang in Wei's device. This figure shows two screen shots that capture the interactions between two ME devices Chen and Wei.

7. Summary

This paper proposed iMobile, a proxy-based platform for developing mobile services for various mobile devices and wireless access technologies. iMobile introduces three abstractions on top of an agent-based proxy: dev-let that interacts with various access devices and protocols; info-let that accesses multiple information spaces; and app-let that implements application and service logic. The let engine arbitrates the communications among dev-lets, app-lets and info-lets, which also maintains user and device profiles for personalized services. The iMobile vision allows a mobile user to access vast amounts of information available on various wired and wireless networks regardless of where the user is and what device or communication protocol is available. This modular architecture allows developers to write device drivers, information access methods and application logic independently from each other.

We also developed a simplified iMobile platform called iMobile ME. The iMobile ME architecture provides a uniform architecture on mobile devices, which allows these devices to both communicate with and access resources from each other. As mobile devices become more powerful, iMobile ME provides an ideal infrastructure to facilitate P2P mobile computing.

As a final remark, we have started to integrate iMobile with location services to further eliminate the parameters (zip code, longitude, latitude etc.) that

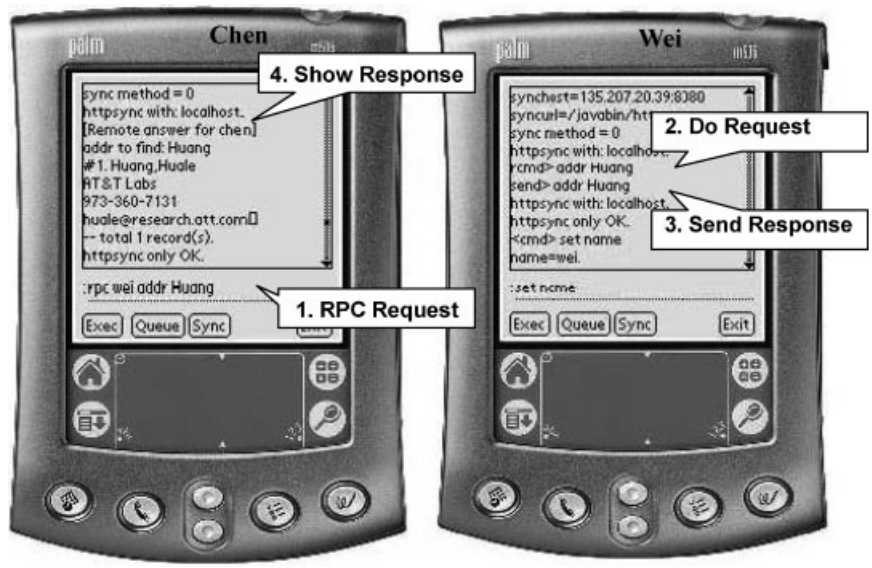


Fig. 23. A remote procedure call example.

a mobile user should provide to access useful information (nearby restaurants and hospitals etc.). We are also experimenting with Voice XML technologies to support a voice-based dev-let for information retrievals.

Acknowledgement

The iProxy and iMobile platforms are research projects funded by AT&T Labs Research. We thank Robin Chen, Josie Cheng, Di-Fa Chang and all the persons who participated in the projects. This work was partially funded by NSC Program for Promoting Academic Excellence of Universities.

References

1. AltaVista Company. AltaVista discovery. http://discovery.altavista.digital.com/, 1998.
2. American Online, Inc. AOL instant messenger. http://www.aol.com/aim, January 1999.
3. The Apache Group. The Apache Cocoon Project. http://xml.apache.org/cocoon/index.html, 2000.
4. Fox A, Gribble SD. Security on the move: indirect authentication using Kerberos. In *Proceedings of Second ACM Conference on Mobile Computing*, White Plains, New York, November 1996.
5. Fox A, Gribble SD, Chawathe Y, Brewer EA. Adapting to network and client variation using active proxies: lessons and perspectives. *IEEE Personal Communications*, 1998; **5**(4): 10–19.
6. AT&T. Your WorldNet. http://yourworldnet.planetdirect.com/, January 1999.
7. AT&T. Wireless data services. http://www.att.com/pocketnet/, 2000.
8. AT&T Labs. iProxy. http://www.research.att.com/sw/tools/iproxy/, January 1999.
9. BlackBerry. Wireless solution. http://www.blackberry.net/product/blackberry/.
10. Myers BA. The Pittsburgh Pebbles PDA Project. http://www.cs.cmu.edu/∼pebbles/.
11. CompanionLink. http://www.companionlink.com/.
12. European Telecommunications Standards Institute. Use of DTE-DCE interface for short message service (SMS) and cell broadcast service (CBS) (GSM 07.05 version 5.5.0). http://www.esti.org/, January 1998.
13. Extended Systems. http://www.extendedsystems.com/.
14. Douglis F, Ball T, Chen YF, Koutsofios E. The AT&T internet differencing engine: tracking and viewing changes on the web. *World Wide Web Journal* 1998; **1**(1): 27–44. (Baltzer Science Publishers).
15. Gnutella. http://www.gnutella.com/.
16. GSM Association. An overview of SMS. http://www.gsmworld.com/technology/sms.html.
17. Wang H, Raman B, Biswas R, Chuah CN, Gummadi R, Hohlt B, Hong X, Kiciman E, Mao Z, Shih J, Subramanian L, Zhao BY, Joseph A, Katz R. ICEBERG: an internet-core network architecture for integrated communications. *IEEE Personal Communications*: Special Issue on IP-based Mobile Telecommunication Networks, 2000.
18. Rao H, Chang DF, Lin YB. iSMS: an integration platform for short message service and IP network. *IEEE Network* 2001; **15**(2): 48–55.
19. Rao H, Chen YF, Chang DF, Chen MF. iMobile: a proxy-based platform for mobile services. *The First ACM Workshop on Wireless Mobile Internet* (*WMI 2001*), Rome, July 2001.
20. Rao H, Chen YF, Chen MF, Chang J. iProxy: a programmable proxy server. In *Poster Proceedings of the WebNet99 Conference*, October 1999.
21. Rao H, Chen YF, Chen MF, Chang J. A proxy-based personal portal. In *Proceedings of the WebNet99 Conference*, Hawaii, October 1999.
22. Rao H, Chen YF, Chen MF. *A proxy-based web archiving service*. In *Proceedings of the Middleware Symposium*, Portland, Oregon, July 2000.
23. Jabber. http://www.jabber.org/.
24. Steiner J, Neuman C, Schiller J. Kerberos: an authentication service for open network systems. In *Proceedings of the Winter 1988 Usenix Conference*, February 1988; pp. 191–201.
25. Myers J, Rose M. Post office protocol—version 3. *RFC1939*, May 1996.
26. Aberer K, Hauswirth M. Peer-to-peer information systems: concepts and models, state of the art, and future systems. *18th International Conference on Data Engineering*, San Jose, February 2002.
27. Crispin M. Internet message access protocol. *RFC2060*, December 1996.
28. Nilsson M, Hjelm J, Ohto H. Composite capabilities/preference profiles: requirements and architecture. http://www.w3.org/Mobile/CCPP/, W3C working group, 2000.
29. Napster, Inc. http://computer.howstuffworks.com/napster1.htm.
30. OmniSky. CDPD modem for PalmV. April 2000.
31. OMG, Inc. CORBA: common object request broker architecture. http://www.corba.org/.
32. Palm, Inc. Web clipping. http://www.palmos.com/dev/tech/webclipping/.
33. Barret R, Maglio PP. Intermediaries: new places for producing and manipulating web content. In *Proceedings of the Seventh International World Wide Web Conference*, Brisbane, Australia, 1998.
34. SmartHome, Inc. X-10/PLC products. http://www.x10.org/.
35. Wildstrom SH. Should coffeepots talk? *Business Week* 1999: 22.
36. Czerwinski SE, Zhao BY, Hodes TD, Joseph AD, Katz RH. An architecture for a secure service discovery service. In *Proceedings of The Fifth ACM/IEEE International Conference on Mobile Computing (MobiCom'99)*, Seattle, WA, August 1999; pp. 24–35.
37. Sun Microsystems. Java 2 micro edition. http://java.sun.com/j2me/.
38. SyncML Initiative Ltd. SyncML. http://www.openmobilealliance.org/syncml/.
39. Varshney U. Recent advances in wireless networking. *IEEE Computer* 2000; **33**(6): 100–103.
40. The WAP Forum. Wireless application protocol. http://www.wapforum.org/.
41. Wedgetail Communications. JCSI (Java crypto and security implementation) micro edition. http://www.wedgetail.com/jcsi/microedition/.
42. Fu XD, Shi WS, Akkerman A, Karamcheti V. CANS: composable, adaptive network services infrastructure. *USENIX Symposium on Internet Technologies and Systems (USITS)*, March 2001.
43. Yahoo, Inc. My Yahoo. http://my.yahoo.com/, January 1999.
44. Wang YM, Russell W, Arora A. A toolkit for building dependable and extensible home networking applications. In *Proceedings of 4th USENIX Windows Systems Symposium*, August 2000.
45. Chen YF, Koutsofios E. WebCiao: a website visualization and tracking system. In *Proceedings of WebNet97*, Toronto, Canada, October 1997.

Authors' Biographies

Ming-Feng Chen received the Ph.D. in computer science from the National Chiao Tung University, Taiwan, in 2004. He was the research consultant in AT&T Labs Research (1995–2001). His primary research interests are mobile computing, wireless and data communication, proxy architecture and internet services.

Yi-Bing Lin received his B.S.E.E. degree from National Cheng Kung University, Taiwan, in 1983, and his Ph.D. in computer science from the University of Washington in 1990. From 1990 to 1995, he was with the Applied Research Area at Bell Communications Research (Bellcore), Morristown, NJ. In 1995, he was appointed as a professor in the of Department of Computer Science and Information Engineering (CSIE), National Chiao Tung University (NCTU). In 1996, he was appointed as deputy director of Microelectronics and Information Systems Research Center, NCTU. During 1997–1999, he was elected as chairman of CSIE, NCTU and is now chair professor of NCTU. His current research interests include design and analysis of personal communications services network, mobile computing, distributed simulation and performance modeling. Dr. Lin has published over 150 journal articles and more than 200 conference papers.

Herman C.-H. Rao is the vice president in Far EasTone Telecommunication Co. Ltd. in Taiwan, responsible for developing mobile technologies, telephony and data platforms, and products. He led a team that successfully developed Service Platform in 2001, the first mobile service platform enabling Mobile Internet Business, and launched multi-media portal and products in 2003. In addition, Herman established Far Eastone Labs in 1998 and since leads the Labs, that research advanced telecommunication technologies to support Far EasTone as a multiple-function wireless communication operator in Taiwan. Dr. Rao has a doctorate in computer science from the University of Arizona, and a B.S. in mechanical engineering from National Taiwan University. Prior joining Far EasTone, he worked in Bell Labs and AT&T Research as a senior researcher for 10 years. He holds four U.S. patents and published more than 50 articles in conferences and journals. In additional to extensive industry experiences in telecommunication and data communication, Dr. Rao was an associate professor in National Tsing-Hua University and National Chung-Cheng University.

Quincy Wu received his B.S. degree in mathematics from National Tsing Hua University, Taiwan, in 1992, and his Ph.D. in computer science and information engineering from National Tsing Hua University in 2000. He joined National Center for High-Performance Computing with the NBEN (National Broadband Experimental Network) project, where he successfully designed and established the first island-wide IPv6 network among universities. In 2002, he began serving as a research assistant professor with National Chiao Tung University, and helped National Telecommunications Project Office to deploy a SIP-based VoIP Platform across several universities. His current research interests include session initiation protocol, open service architecture, internet protocol version 6, design and analysis of approximation algorithms and service creation on the third generation mobile network.