The Satchel System Architecture: Mobile Access to Documents and Services

Mike Flynn[a], David Pendlebury[b], Chris Jones[c], Marge Eldridge[a], Mik Lamming[a]

[a]*Xerox Research Centre Europe, 61 Regent Street, Cambridge CB2 1AB, UK*
[b]*Aspect Capital Limited, 9 Mandeville Place, London W1M 5LB, UK*
[c]*Xerox Mobile Solutions, Burleigh House, 13-15 Newmarket Road, Cambridge CB5 8EG, UK*

Mobile professionals require access to documents and document-related services, such as printing, wherever they may be. They may also wish to give documents to colleagues electronically, as easily as with paper, face-to-face, and with similar security characteristics. The Satchel system provides such capabilities in the form of a mobile browser, implemented on a device that professional people would be likely to carry anyway, such as a pager or mobile phone. Printing may be performed on any Satchel-enabled printer, or any fax machine. Scanning, too, may be accomplished at any Satchel-enabled scanner. Access rights to individual documents may be safely distributed, without regard to document formats. Access to document services is greatly simplified by the use of context sensitivity. The system has been extensively tested and evaluated. This paper describes the architecture of the Satchel system.

1 Introduction

In this paper, we describe the architecture of a system called "Satchel". Satchel supports mobile document work by providing streamlined access to remote documents, via a mobile browser running on a Nokia 9000 Communicator. In addition, Satchel provides a set of specialised document services that are required to support mobile document work such as printing, faxing, and scanning. The current mobile browser uses both infrared and GSM radio to communicate wirelessly with the rest of the Satchel system.

In a companion paper [Lamming et al., in press], we review the distinctive features of mobile document work, and explain how they motivated us to formulate five design goals for any system seeking to provide effective support. To summarise, these goals are:

(a) **Easy access to document services**, providing a simple way to locate and use document services such as printing, faxing or scanning. A document service might be associated with a physical device (e.g., a print service), or it might not (e.g., an e-mail service).
(b) **Timely document access** for any document when and where it is required.
(c) **A streamlined user interface** that achieves high usability through specialisation for a limited set of activities.
(d) **Ubiquity**, since mobile document work occurs in many different contexts.
(e) **Compliance with security policies**, providing a level of security for documents and document services that meets or exceeds policies set by the individual or the organisation.

Our companion paper discusses how Satchel addresses these design goals in general. In this paper, we describe the architecture of the Satchel system in more detail, referring to how specific aspects of the design address these goals.

The next section (§2) describes the background and some early Satchel prototypes. The overall design of the current Satchel architecture is described in §3, then in more detail in the following four sections. The proprietary format used in transmission to the browser is described in §8, while security measures are described in §9, and a deployed system is described in §10. Some remaining challenges and possible future developments are described in §11.

2 Background and Prototypes

2.1 The Seed: Forget-Me-Not (1993)

Forget-Me-Not [Lamming and Flynn, 1994] was a context-sensitive retrieval system based on principles of episodic memory and ubiquitous computing [Weiser, 1991]; it used the Xerox PARC Tab [Want et al., 1995] as the mobile client. In Forget-Me-Not, information was automatically collected about a person's activities: for example, telephone calls made, e-mail messages received, and rooms visited. This information was logged in a central database, and presented as a personal biography on the PARC Tab screen, with activities presented in chronological order. The biography could be browsed and filtered to show only a particular type of entity or activity. So, for example, a user could manipulate icons on the PARC Tab screen to answer questions such as: "Which

To appear in: Mobile Networks & Applications (MONET).

document did I print just before the meeting with Bob last month?"

Included in the logged information were document-related events, such as passing documents to one another or printing them. Because Forget-Me-Not could determine information about the user's context, such as the people and equipment in the room, it became natural to think about transferring documents to others, or printing them, merely by dragging and dropping the appropriate icons on the PARC Tab screen. This was the seed of the Satchel idea: to take advantage of the user's context to streamline the user interface for printing and transferring documents.

2.2 Early Satchel Prototypes

During the lifetime of the Satchel project, we have developed several prototypes of the system on various mobile and desktop computing platforms. These prototypes are described below.

The first Satchel prototype was implemented using the PARC Tab as the mobile client and an Apple Macintosh computer. The PARC Tab screen showed the contents of a Satchel folder on the user's Macintosh desktop. As the PARC Tab was carried around, people (other Satchel users) or equipment encountered also appeared on the Tab screen as small icons. To transfer a document to a person, the user simply dragged and dropped the document's icon over the person's icon; the document would immediately appear in the other person's Satchel folder, and hence on their screen. Similarly, to print a document on a nearby printer, the user simply dragged the document icon over the printer icon. Although this crude prototype was flawed (e.g., dragging a person to a printer printed the contents of that person's Satchel folder on the Macintosh), had no security, and relied on local PARC Tab infrastructure, it was sufficient to stimulate further work on the Satchel project.

The next series of prototypes was implemented on the Apple Newton PDA [see McKeehan and Rhodes, 1996]. At this time, the Newton was widely available, relatively cheap, and easy to program. Communication was via infrared, serial cable or an AppleTalk network, and local radio communications cards became available later.

In early 1994, a very basic Satchel browser allowed the user to navigate around his or her UNIX directories. The user could carry out one of three simple operations on files and directories: BLOW, SUCK or SYNC. These commands were formulated in the mobile 'roaming' Newton and passed over infrared to one of many static 'tethered' Newtons, placed by printers, monitors or scanners. BLOW indicated that the selected document should be passed to the receiver (e.g., a printer) or passed to another Satchel user. SUCK indicated that a new entry should be made in the current directory, and that the device should place a document into that location, by scanning, for example. SYNC caused the browser information to be updated from the UNIX system (see §11.4 for detail). Note that the Newton never contained the documents, but merely issued commands referring to them. The wired infrastructure carried the actual documents for printing, and so on.

By mid-1994, the World Wide Web was becoming prevalent, and the Newton Satchel system was modified to use generic Web requests rather than the custom UNIX command repertoire. The benefits of basing Satchel on Web protocols included the large and growing infrastructure, ready availability of many components, such as web servers and scripting libraries, and the ability to test most system functionality using a standard Web browser such as Netscape Navigator. The rapid growth of the World Wide Web also meant that the problems of scaling Web-based services to improve capacity, reliability and security were being addressed by many people.

By the end of 1995, the Satchel infrastructure was re-implemented under Microsoft Windows NT, and the browser's user interface was greatly simplified. The replacement of SUCK, BLOW and SYNC, with an all-purpose DO-IT command, completely eliminated the need for the user to know which operation was appropriate in which physical circumstance. DO-IT was interpreted by the receiver, invoking the most appropriate action for the given context. For example, in front of a printer, the selected document was printed; in front of a display screen, it was displayed; in front of a person, the document (or rather a reference to it) was transferred. In front of a scanner, a directory was selected, and the document in the scanner was scanned and stored under a default name, based on the location, date and time. Also, familiar Web-browser buttons, such as HOME, BACK and RELOAD were introduced.

An additional ROOM SERVICE button asked the local infrastructure what services were available at that place. Often, the page returned would be a form for invoking a specific service, such as printing, with various options. In the case of multifunction machines, a directory of such service forms could be returned.

By July 1996, to help us achieve our security goal, extra mechanisms were added. Each document reference now incorporated a digital signature [Salomaa, 1990], which proved that the reference had indeed been generated by the Satchel claiming to have done so (§9). In addition, a local radio system covering our entire laboratory was added to increase the ubiquity of the Satchel system.

Although the Newton-based prototype was effective, the local radio system was only of use within our laboratory. Outside the building, a mobile phone, modem card and cable were required to communicate. Around that time, the Nokia 9000 Communicator, a mobile phone with built-in programmable PDA, infrared and GSM data communications, became available. Subsequent browser development used this device.

3 Architectural Overview

The current Satchel system architecture, illustrated in Figure 1 overleaf, draws upon our experience with the prototypes discussed in §2. The Satchel system consists of

a set of browsers, gateways, servers and services, which treat the local Intranet and the Internet as a file system. Initially, we will discuss use with the Intranet, and later extend to the Internet.

A Satchel Browser (§4) provides a user interface on the mobile client for navigating to documents and invoking services on them. While a cache mechanism minimises the amount of communication required between the browser and the rest of the system, a browser may communicate to explore new directories, invoke services, or refresh the cache. To minimise transmissions over relatively low-bandwidth connections, Satchel Browsers do not deal directly with documents, but with secure document references, or *tokens*. A token may be passed directly to another browser over infrared, or *Beamed*, in order to convey access permission to a specific document. Similarly, tokens may be passed to document services, to grant them permission, say, to fetch the document for printing. Thus, wireless traffic usually consists of service invocations, tokens, or directory listings, whereas services use the wide-band wired network for transferring the documents themselves.

From a user's perspective, services are provided by Satchel-enabled devices, such as printers or scanners. In fact, each device has attached an infrared transceiver, connected to some nearby PC running a simple Gateway (§5) process. The Gateway injects contextual information into passing requests, such that services appropriate to that situation may be offered. This helps to streamline the user interface. In the absence of infrared facilities, a standard connection using Point-to-Point Protocol (PPP) may be made via the GSM cellular radio network to a Satchel server back at base.

While Satchel users need neither know nor care about the implementation, the protocols used in the Satchel system are entirely compatible with the World Wide Web. The extensible nature of the Web's HTTP protocol lends itself to the construction of new systems such as Satchel without sacrificing interoperability with existing infrastructure. For example, normal Web servers may be used to access public or company documents and directories. Existing techniques for load sharing between multiple Web servers, such as those commonly employed at large Web sites, can be used to increase the capacity and availability of the Satchel system.

For more private documents, the Satchel Personal Web Server (§6) behaves just as a normal Web server, but restricts access, such that only the owner, or other legitimate token holder, will gain access to the document, thus enhancing security (§9). Personal Web Servers may also act as hosts for a range of document services, as may any standard World Wide Web server.

The Satchel Services instigate document transactions, such as printing or scanning, and report on their success. Actually, mobile directory and document access is also performed indirectly through such a service — the Fetch Service (§7.1) — capable of translating HTML [Berners-Lee and Connolly, 1995] content into a proprietary condensed format, known as Halibut (§8), for efficient transmission and display on the browser.

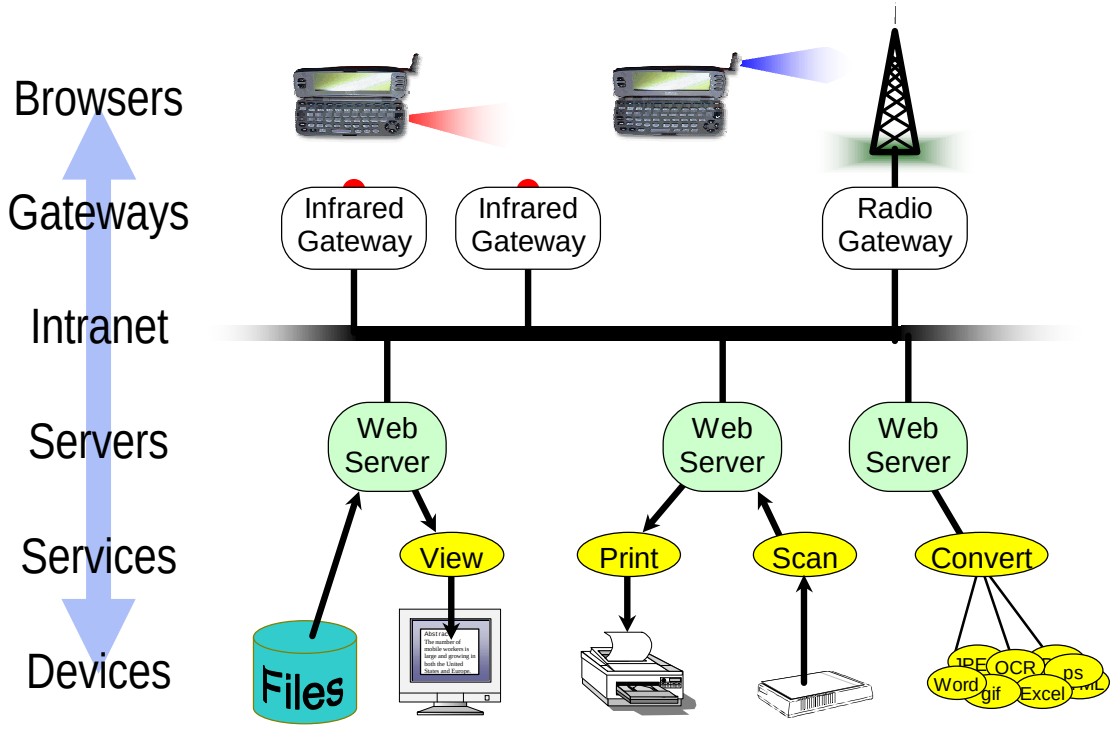


Figure 1. The Satchel System Architecture

4 The Browser

A Satchel Browser must provide the ability to browse directories for documents, and to invoke services upon them, once found. Browsers may be unable to display documents themselves, for reasons of transmission bandwidth, storage requirements, screen size and resolution. For example, one experimental browser is implemented on the PARC Minder, a tiny pager-sized device with only a two-line screen and infrared communications, illustrated in Figure 2 below. Omission of unnecessary features helps to streamline the user interface, one of our design goals.

Figure 2. The PARC Minder Satchel Browser

The current Satchel Browser is essentially a mobile World Wide Web browser, implemented on the Nokia 9000 Communicator, a mobile phone incorporating a sophisticated PDA (see Figure 3 below). Like any normal Web browser, the Satchel Browser provides:

?? selection and following of hyperlinks, to other HTML pages or directories;
?? Home, Back and Reload buttons, operating in the usual manner;
?? nomination of a 'home' page, typically an HTML page on a user's workstation, with links to useful places within the user's private filing system, or the Internet;
?? nomination of a home server and proxy server for radio connections (infrared connections use a server nominated by the Gateway);
?? fully featured form handling (but currently not multiple forms per page);
?? considerable caching of pages (around 1000 pages in Halibut), purged on a least-recently-used basis, which is especially useful given the latency and expense of radio network connections.

These features impart familiarity to the technically aware user. There are, however, several significant differences between a Satchel Browser and a conventional Web browser. These differences concern browsing, security, services and Beaming, and are discussed below.

4.1 Browsing

When browsing Web pages with the Satchel Browser, the user selects a link with the cursor, and presses the OPEN button (see Figure 4 below). If the page is not already cached, then an HTTP [Berners-Lee et al., 1996] request must be made, over a communications link, to a Web server hosting Satchel services. Communication is normally attempted first over infrared and then over radio (subject to user preferences). If infrared contact is made,

Figure 3. The Nokia 9000 Communicator

the receiving infrared gateway forwards requests to its nominated server — after all, this might be unfamiliar territory, well away from home, where the local services are not known to the user. With radio, however, the browser must nominate a known server, since the connection is typically via a general PPP connection to an ISP, or directly into a company Intranet.

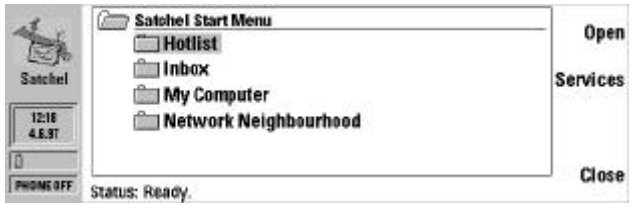


Figure 4. The Satchel Browser on the Nokia 9000 Communicator

All requests for a document, or a directory listing, issued by the browser are made indirectly through the Fetch Service (§7.1). Whenever the browser makes any request, it inserts an HTTP 'Accept' header to inform this service that the browser only accepts Halibut replies, and it arranges for translation of replies into Halibut. Without this header, no translation to Halibut occurs, allowing the services to be tested from a regular Web browser by means of a regular HTML form, expecting HTML in reply. In any case, future browsers might use other formats or even HTML itself, which will be indicated by this header.

The browser expects to receive Halibut, in which there is no provision for images or sound. If the retrieved document was not in HTML, and therefore cannot be translated to Halibut, the Fetch Service replies with details of the file in question (still in Halibut, so that the browser always receives the correct format). This prevents potentially large images, movies or other non-HTML files, from being transmitted over the low-bandwidth link, only to be rejected at the far end by the browser. Fortunately, Web servers commonly serve up directory listings in

HTML, which are translated to Halibut, just like any other document.

4.2 Service Invocation

A SERVICES button (see Figure 4 above) provides easy access to document services, appropriate to the context, another of our design goals. Typically, a user browses to a document of interest, and wishes to invoke some service on that document — to print it on a nearby printer, for example. When in front of a Satchel-enabled device, the user presses the SERVICES button, which makes an infrared request for services (§7.2) available at that location, and expects back an appropriate service form or a directory of such forms, where multiple services are possible. Once again, the request uses standard HTTP, with a header indicating that only Halibut is accepted in reply, just as with fetch requests.

In the case of a radio request for services, no contextual information is known. Exactly the same service request is sent to the nominated home server, devoid of location information. The reply is typically a directory of services available at the home site, such as printing or faxing.

The service forms are just normal HTML Web pages and forms, arranged by local system administrators, which, once again, are translated to Halibut on the fly, and sent to the browser. The browser supports menus, fields, checkboxes and other form elements. Typically, they contain fields to be filled in with the document's title, user's name and other options, such as number of copies. They can be visited and used with a regular Web browser for testing purposes, although hidden fields (discussed below) may need to be made visible.

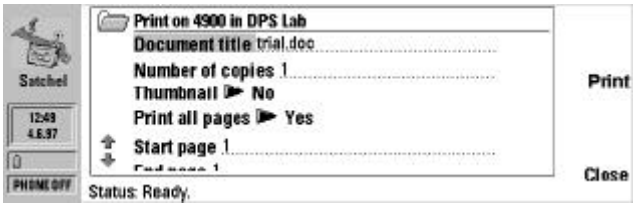


Figure 5. A typical Print Service form.

The browser treats such forms, such as that in Figure 5 above, just like any other form found on the Web, but, knowing that the SERVICES button was pressed on an ancestor, enables a few extra features:

1) Fields with well-known names may be filled in automatically by the browser; for example, the 'SATCHEL_USER' field is filled in with the user's name, and the 'SATCHEL_TOKEN' field is filled in with the token of the document originally selected. Such fields might be hidden from the user to avoid clutter and accidental change, if specified in the form's original HTML.
2) The first button occurring on the form is promoted out of the form itself, to replace the (disabled) SERVICES button. This syntactic convenience allows common operations, such as printing with default options, to involve simply pressing the same physical button twice. The effect is similar to that of the DO IT command in the Newton Satchel Browser (§2.2), but with greater flexibility, as the user can select service options, if required.
3) The visual appearance of the forms obtained through the SERVICES button may be changed, to appear as dialog boxes rather than conventional pages, keeping the selected document in view in the background. This is not currently implemented, due to difficulties with the Nokia 9000's operating system (GEOS).
4) The CLOSE button now takes the browser back directly to the condition in which the SERVICES button was originally pressed, rather than via any previous service page, in contrast to behaviour during browsing.

Pressing the form's operation button packages up the form contents as an HTTP POST request, and transmits it as the argument to a Fetch Service request. Typically, one of the (hidden) fields of the form contains the document token of interest. When the relevant service is invoked, it unpacks its arguments to find the token, and uses that to fetch the actual document. Each document service replies with a page describing success or failure, possibly with a SHOW LOG button for further detailed service information. (Since this page is still derived from the original SERVICES button press, this SHOW LOG button is now in place of the browser's SERVICES button.)

4.3 Beaming Tokens

The browser can 'Beam' a token to another browser, over infrared. This is used in casual encounters or in meetings to distribute documents quickly, or, more precisely, to distribute references with access rights.

Each browser offers a service, the Beam Service (§7.7), that receives beamed tokens. The browser must maintain a page of such tokens, called the Inbox. This page is accessed through a special Inbox URL, so that, for example, the user's home page can contain a link to it, despite the fact that it really resides in the browser. We consider this preferable to using a dedicated hardware button, since these are in short supply on small devices.

Tokens received by the Beam Service are in a somewhat vulnerable state, in that they might exist only within the mobile device. In order to secure them, they should either be sent to the workstation desktop with the View Service (§7.4), sent home using the E-mail Service (§7.9), or the documents they represent should be printed. In the future, we envisage there will be a mechanism for automatically synchronising the Inbox with a workstation folder, and informing the user of the status of each Inbox token.

5 Gateways

A Gateway is a simple process, which links the infrared medium to that of the local area network. It listens to the infrared transceiver associated with a Satchel-enabled device, waiting for requests from browsers, and forwards

them on to a Web server hosting Satchel services, over TCP/IP. However, the infrared Gateway also inserts context information into the request as it passes through, as extra HTTP headers. Data returned by a service is transmitted straight back through to the waiting browser.

Each Gateway process has a simple user interface for specifying which infrared transceiver to listen to, which Web server to forward to, and what context information to insert. Many Gateways may use the same Web server, the context information being used to determine the originator. This information is flexible, and may be managed according to local requirements. Typically, it identifies the location, the name of the device, and the service to offer. Whilst the fetch service ignores this information, service enquiries use it to determine which service form to reply. Although the device and service could be determined from the location information alone, this would require some central database to perform that mapping.

Infrared communications take place using a simple proprietary protocol, originally developed for the PARC Tab and Minder, rather than IrDA [IrDA, 1997]. This was due in part to the desire for compatibility between prototypes, but mainly due to lack of a suitable IrDA protocol stack for Windows NT at that time.

Radio connections over dial-in lines, possibly through an outside ISP, use PPP to obtain Internet connectivity. With infrared, the Gateway nominated a Web server to use on the browser's behalf, whereas with generic PPP connections the browser itself must make that nomination. Typically, a well-known server is used, since in this situation the network environment is known. In the case where an outside ISP is being used, a special proxy server on the firewall must also be nominated in order to cross it. This proxy authenticates passing requests (see §9), possibly denying access.

Other communication methods may have their own Gateways. For example, our Newton-based prototype used a radio card to make AppleTalk connections to our local network. A bridging process relayed messages to a Web server over IP and back again, just as the current infrared Gateway does. We envisage other Gateways to paging networks and the e-mail server infrastructure, too.

6 Servers

The native filing system for Satchel is the World Wide Web — our file servers are just Web servers. The security design goal required that a user should not need to publish their private documents on the World Wide Web in order to access them when away from the desk. Also, we believed it to be important to present users with the same familiar organization of documents available from their workstation. Unfortunately, a typical Web server works at the level of directories and files, with a fixed set of "document roots", and is incapable of providing access to other machines, directories or files expected by the user (such as "Network Neighbourhood" or "My Computer" in Windows NT).

To overcome these problems, at least in the Windows NT environment, we developed the Satchel Personal Web Server. Satchel users are given access to the directory listings and files that they have permission to access from their desktop PCs – no more, and no less. Access is controlled by means of the Satchel security mechanisms (§9).

An obvious benefit of using Web servers as file systems is the potential to access documents residing on any regular Web server (on a corporate intranet or the Internet) or any electronic file system or document repository that has a web front-end (e.g., Xerox DocuShare, Lotus Notes, etc.).

7 Services

Satchel services are invoked directly and indirectly by Satchel Browsers. In fact, a browser's only contact with the world is through these services.

Services are implemented as a set of CGI scripts [NCSA], hosted by a regular Web server, or a Satchel Personal Web Server. Invocation is by HTTP request to the server, so that any Web browser can invoke these services for testing purposes, in principle, given an appropriate form.

A CGI script qualifies as a Satchel service by adhering to a number of simple conventions. These include defined input parameter names, error reporting and logging mechanisms, and output formats. Given details of these conventions, a Satchel service can be implemented using any Web server development tools (such as shell scripts, Python scripts and Java servlets). To ease the implementation of new services, a Python class hierarchy and library were developed which essentially provided skeletal Satchel services, with methods that may be overridden to customise functionality. Making such a new service available to users simply requires writing an HTML form, following the naming conventions, representing the service's user interface, and adding this form to the directory of services. If this service is associated with a specific device, this association is made through the device's Gateway interface. Otherwise, generic services are linked into the Web page for radio service requests.

Two fundamental services, the Fetch Service and Service Enquiries described below, reply in HTML — or Halibut if the request includes an HTTP 'Accept' header indicating that only Halibut is accepted in reply. The HTML to Halibut converter (§8) is used to do this on the fly, and the much-reduced result is transmitted back to the browser. All other services are invoked via the Fetch Service, and so reply HTML for possible translation.

The Web server logs each service invocation, and services create their own logs in addition, providing a link or button to access them in their replies. This not only aids debugging, but also allows the user to monitor the progress of long-running service requests.

The following sub-sections describe those services available in the current Satchel system. Note that some of these services are not directly visible to the user.

7.1 The Fetch Service

While transparent to the end user, the Fetch Service underpins browsing and service invocation (§4.1). This service is used to retrieve HTML pages, including directory listings and service forms, from the World Wide Web and translate them to Halibut. It takes a token for the required page as argument — lexically just an HTTP request — and contacts the appropriate Web server or proxy, if necessary, to obtain it. It is also used to retrieve documents by the other services, in which case no translation occurs.

7.2 Service Enquiries

This enquiry service yields a form for the service provided at the given location. Where more that one service is available, a directory of such services might be provided. The location information is supplied by the infrared Gateway (§5). In the case of radio communication, no location information is available, and so a full directory of available services is returned. This might include the Fax, Print and E-mail services described below.

A service directory might be organised by function, location, or indeed any other appropriate organisation, at the discretion of the local system manager. The list of available services might even be dynamically determined from all available information. This Service Enquiry mechanism makes simple, effective use of the locality of infrared communications. More complex systems, such as that of Baggio and Piumarta [1996], use a relatively sophisticated distributed object and remote procedure call mechanism to achieve little more.

7.3 The Print Service

The Print Service fetches and prints documents, virtually regardless of their type. The service takes as argument the printer to use, various printing options and a token for the document to be printed. The service fetches the document, noting its MIME type. Then, Windows information regarding the indicated printer is examined to determine what formats the printer accepts. To convert between these types, the Conversion Service (§7.6) is invoked, and the result is sent to the printer.

As with many services, this whole process may be lengthy, and so fetching conversion and printing are performed asynchronously — that is, the Print Service actually returns an immediate acknowledgement, but which contains a link to the Log of the on-going print process.

7.4 The View Service

The View Service presents a document on a Satchel-enabled monitor or display screen. As with the Print Service, the document represented by the token argument is fetched and the MIME type noted. Then, Windows information regarding the locally available display applications is gathered[1]. The Conversion Service (§7.6) is invoked, if necessary, and the resultant file is opened on the display. Note that this is a local copy of the document, possibly in a different format to the original.

Tokens can be held in a PC file, rather like Windows Internet Shortcuts. An extra argument to the View Service chooses whether to just put the token on the desktop like this, and defer the fetching, conversion and display. Such token files have a type and application associated with them, such that, at any later time, the token can be opened with the mouse, causing the above process to occur. Saving tokens in this way is particularly useful with tokens that have been collected from other Satchel users by the Beam Service (§7.7), which are not held elsewhere.

7.5 The Scan and Squirrel Services

The Scan and Squirrel Services work together to allow paper documents to be scanned, potentially at any Satchel-enabled scanner in the world, with the results arriving back on the user's workstation, or other selected location.

Since no token exists for the as yet unscanned document, the user must select a directory, into which the document will be scanned, press the Services button, and obtain a Scan Service form. This form contains a default name field for the scanned file, which may be edited in the Browser. Upon submission of the form, by pressing the Scan button, the document is scanned and the named file arrives in the directory originally selected. This process is implemented in several phases:

1) The local Scan service invokes the Squirrel Service on the remote machine, where the file should be created, to check for permission and to reserve the name. The Squirrel Service is passed the signed token for the directory of choice, and the file name required, as arguments.
2) The remote Squirrel Service attempts to create a placeholder file of the given name in the given directory[2]. This file happens to contain an explanatory message, should it be opened unexpectedly. The Squirrel Service replies to the Scan Service indicating success or failure.
3) Assuming success, the Scan Service causes the scanning device to scan the document to a local temporary file.
4) The remote Squirrel Service is invoked again, but given a token for the local temporary file.
5) The Squirrel Service fetches the temporary file contents and overwrites the placeholder file with them.
6) Finally, the local temporary file is deleted.

[1] Under Windows NT, applications cannot easily be started on remote machines. Therefore a Web server and the View Service must run on the PC hosting the display, invoked from the Gateway's nominated Satchel Server.

[2] The Squirrel Service should check the signature of the requestor, but currently does not.

Thus, file scanning and storage to a remote machine is accomplished, without recourse to the HTTP PUT method or file upload mechanism, both of which were poorly supported by the World Wide Web infrastructure, at least at the time of implementation. This mechanism is especially worthwhile for large documents, since no physical scanning takes place until there is reason to believe that the data can be stored at the remote location, assuming space is available. Despite the relative complexity of this scanning process, from the user's perspective this is as simple as printing or any other such service.

7.6 The Conversion Service

The Conversion Service is not directly visible to the user, but is used by other Satchel Services, such as Print (§7.3) and View (§7.4), to match the available document format with desired formats for rendering in a manner similar to that of System 33 [Putz, 1993].

This service can convert documents between many different formats, using a graph of all the available conversion utilities, between different MIME types [Borenstein and Freed, 1993]. When a request for conversion arrives, the graph is searched for the best conversion route between the given types. Multiple alternative destination types may be specified. The 'best' is currently taken to be the shortest, although a more sophisticated scoring system could be used. Each conversion utility is then run, and the final result returned.

Some conversions use the original application (Microsoft Word to PostScript, for example). Others use free or commercially available versatile conversion utilities. Some single-step conversions work from file to file only, while others can work with streams and may be plumbed together in a pipeline. The graph representation incorporates all this information, and the service is flexible enough to accommodate them.

The graph is kept in an editable file, and can easily be extended to include new MIME types and conversion utilities. Faced with some new or obscure MIME type, any utility, which can convert it to some other accommodated type, and vice-versa, will link the new type into the graph, so allowing Satchel to work with documents of this type. Currently, the graph has around thirty types (graph nodes), and sixty 'atomic' conversions (graph edges), leading to hundreds of end-to-end conversion possibilities.

Some conversions appear unlikely, but do work. For example, a raster image (such as a Windows bitmap) can be converted to plain text, since ScanSoft's Textbridge OCR package is just one edge mentioned in the conversion graph.

The consequences of using the Conversion Service are that, for example, when printing a document, the user need not know nor care about the document's type, need not own a copy of the application that created it, nor need know anything about the printer — provided that the Conversion Service can cope[1]. This eliminates the need for everybody in an organisation to install every possible application whose documents they might handle. This feature is particularly useful when given a token for a document of unknown type, or when travelling and wishing to print a document in an unknown environment.

7.7 The Beam Service

Users can exchange tokens using the Beam Service, which is simply a service offered by mobile browsers (§4.3) upon receipt of a Service Enquiry over infrared. One user, the donor, selects a document and presses the Services button in front of another user, the recipient. As usual, the donor transmits a service enquiry, asking what services are available here. The recipient's browser, acting just like a Web server, replies with a Beam Service form, containing two fields: one hidden field for the token of the originally selected document, the other visible for its textual name. The donor's browser fills in this form, using the usual automatic mechanisms. Pressing the promoted Beam button transmits the form contents to the recipient, which finally replies with a confirmation message, just like any other service.

The recipient adds the received token to a database of such tokens, known as the 'Inbox'. As described in §4.3, the token is already signed by its originator — perhaps the donor or whoever beamed it to them. Thus, the token must be used verbatim, without further signing, whenever the recipient uses this token.

7.8 The Do-It Service

The Do-It Service is passed a token as argument, which, together with location information from the infrared Gateway (§5), is sufficient to invoke the default service with default parameters for that location. Thus, in front of a printer, Do-It performs as the Print Service, by a scanner as the Scan & Squirrel Services, by a monitor as the View Service.

This service is no longer used by the Nokia 9000 implementation, since two button presses (Services, followed by the service form's promoted button) accomplish the same task. However, other implementations of the Satchel browser on simpler devices, such as the 3Com PalmPilot and the PARC Minder, still use this service.

7.9 The E-mail Service

The E-mail Service enables a Satchel user to deliver an electronic copy of any Satchel accessible document to anybody with an Internet e-mail account, regardless of whether they are Satchel-aware.

The service takes as argument a token for the document, the recipient's e-mail address and various other options. An e-mail address can be obtained from the

[1] Documents are not scaled to fit the page, as this might not always be desirable.

Nokia 9000's Contacts application, or just be typed in. Options include supplying a subject and message body for the e-mail message and a choice of delivering the document or the token itself, as an attachment. The service constructs an e-mail message, fetching the document if necessary, and uses the Simple Mail Transfer Protocol (SMTP) to submit the message to an Internet SMTP server.

The advantages of attaching a token rather than the document itself include the extra control the recipient has over when to fetch the document (if at all), possible repeated fetches, and the considerable reduction in size of the message, reducing e-mail congestion at servers and gateways. However, the sender must be sure the recipient is Satchel-aware, or can make use of such a token.

7.10 The Fax Service

The Fax Service enables a Satchel user to obtain a paper copy of any Satchel accessible document, without any reliance on any local Satchel infrastructure (e.g. Satchel-enabled printers). Fax machines, together with GSM radio, help us to satisfy our design goals of ubiquity and timely document access.

This service is very similar in operation to the Print Service: it fetches and faxes documents, virtually regardless of their type. The service takes as argument the recipient's fax number, other faxing options (for example a choice of coversheets), and a token for the document to be faxed. The user may enter a fax number by hand, or select a recipient from the Nokia 9000's Contacts application.

The service fetches the document, noting its MIME type. To convert the document into a raster format suitable for faxing, the Conversion Service (§7.6) is invoked. The converted document and routing instructions are then submitted to a commercial software-based fax server. As with the Print Service, an immediate reply is made with a Log button to monitor progress.

8 Halibut

Transmissions to the Satchel Browser could in principle use HTML. However, HTML has a number of drawbacks in the context of the Satchel application:

?? it is verbose, for example the HTML tag <P> indicates a paragraph break, three characters where one would suffice;

?? it is lexically and syntactically overburdened, for example a typical HTML hyperlink has a 15 character overhead, whereas Halibut uses four (although this could be three);

?? images, tables and frames cannot be accommodated in the Browser;

?? ignored or redundant tags need not be transmitted;

?? authors frequently take liberties with quotation marks, angle brackets and required tags.

The Halibut proprietary format is simple and concise, and trivial to lexically analyse, parse and layout. These properties make it quick to transmit over low-bandwidth links, and reduce battery consumption. The Halibut format consists of a number of directives, one per line, followed by a blank line. Each directive consists of a number of tab-separated fields, followed by a newline character. An example of a Fax Service form is shown in Figure 6 below, while the corresponding Halibut translation is shown in Figure 7.

```
<HEAD>
<TITLE>Fax Service</TITLE>
</HEAD>
<BODY>
<FORM ACTION="/satchelServices/faxService.py" METHOD="POST">
<P>
Document title
<INPUT NAME="SATCHEL_TOKEN_TITLE" VALUE="Document">
<P>
Fax number
<INPUT NAME="SATCHEL_FAX_NUMBER" VALUE="">
<P>With a Cover sheet, you may add a Message.
<P>
Cover sheet
<SELECT NAME = "SATCHEL_COVERSHEET">
  <OPTION VALUE = "" SELECTED>No Cover Sheet
  <OPTION VALUE = "XRCE">XRCE Cover Sheet
</SELECT>
<P>
Recipient
<INPUT NAME="SATCHEL_FAX_RECIPIENT">
<P>
Message
<TEXTAREA COLS=50 ROWS=5 NAME="SATCHEL_COVERSHEET_MESSAGE">
</TEXTAREA>
<P>
<INPUT NAME="SATCHEL_SERVICE_MODE" VALUE="asynchronous"
TYPE="hidden">
<INPUT NAME="SATCHEL_USER" VALUE="Test User" TYPE="hidden">
<P>
<INPUT TYPE="SUBMIT" VALUE="Fax">
<INPUT TYPE="HIDDEN" NAME="SATCHEL_TOKEN">
</FORM>
</BODY>
```

Figure 6. An HTML page for a typical Fax Service form

```
H   Fax Service       0
F   Document title    SATCHEL_TOKEN_TITLE  Document
F   Fax number        SATCHEL_FAX_NUMBER
T   With a Cover sheet, you may add a Message.
I   No Cover Sheet
I   XRCE Cover Sheet  XRCE
M   Cover sheet       SATCHEL_COVERSHEET   No Cover Sheet
F   Recipient         SATCHEL_FAX_RECIPIENT
F   Message           SATCHEL_COVERSHEET_MESSAGE       250
X   SATCHEL_SERVICE_MODE SATCHEL_SERVICE_MODE asynchronous
X   SATCHEL_USER      SATCHEL_USER          Test User
B   Fax               POST    /satchelServices/faxService.py
X   SATCHEL_TOKEN     SATCHEL_TOKEN
```

Figure 7. The Halibut translation of the Fax Service form

The first field in each line indicates the kind of directive. Only the first character is significant: `B` for a button, `F` for a field, or `T` for a text directive, for example (see Table 1, below). This allows future versions of Halibut to represent more information about the directive in the first field. The second field in each line is always the text that should be displayed for this directive: the title of the button or menu item, plain text to display, and so on. Any subsequent fields depend upon the particular directive: for example, a default menu selection, or the symbolic name of a type-in field. When reading Halibut, menu item directives are accumulated until a menu directive is read, whereupon a menu is created using the saved-up items.

Table 1. Halibut directives.

Table 1. Halibut directives.

Directive	Field Contents				
Comment	`#`	`[Text]`			
Text	`T`	`Text`			
Heading	`H`	`Text`	`0|1|2|3|4|5|6`		
Name	`N`	`Text`	`BaseHref`	`[ReloadHref]`	
Link	`L`	`Text`	`Href`	`[Token]`	
Item	`I`	`Text`	`Value`		
Menu	`M`	`Text`	`Name`	`[SelectedText`	`SelectedValue]`
Button	`B`	`Text`	`GET|POST`	`Href`	`[Name]`
Field	`F`	`Text`	`Name`	`[Value`	`[Size|readonly]]`
Checkbox	`C`	`Text`	`Name`	`Value`	`[Checked]`
Hidden	`X`	`Text`	`Name`	`[Value]`	
Rule	`R`				
Error	`E`	`Text`			

Key:		
	`Text`	any ASCII text, excluding control characters
	`Href`	a relative or absolute URL
	`BaseHref`	the base URL for relative Hrefs
	`ReloadHref`	the URL to use to reload this page
	`Token`	a URL-encoded satchel token
	`Name`	the symbolic name of this item
	`SelectedText`	the name of the default selection
	`SelectedValue`	the value of the default selection
	`Checked`	if present (any value) the checkbox is initially checked
	`Size`	a number advising number of characters expected
	[…]	optional
	… | …	alternatives

To make the Halibut format extensible, unknown Halibut directives are ignored (except that their text may be shown), and superfluous fields are ignored, without warning.

Attributes are position encoded by fields, and there is no concept of a closing tag to match. The result is easily machine readable, with very little lexical analysis or syntax checking required. Since text is associated with most directives, layout becomes quite simple too, and can begin immediately upon receipt, reducing latency. Text in Halibut is always followed by a tab or carriage return, so browsers may choose to null terminate received text fields in situ, easing memory management in small devices.

The translator is a stand-alone C program, written using Flex and Bison [Donnelly and Stallman, 1995] to yield a very forgiving lexical analyser and parser, tolerant of missing quotation marks, angle-brackets, ‘required’ tags and so on. This approach of parsing HTML in order to reduce its content for transmission is similar to the more recent Wingman proxy-based Web browser [Fox et al., 1998], though we have not compared results. The conversion to Halibut is lossy: only the limited set of common HTML tags that can be rendered by the browser are converted. The converter reduces the size of a typical HTML page by a factor of two or three (ignoring images), though somewhat less for HTML directory listings due to their simple structure. Currently, the translator does not cope elegantly with multiple forms per page, or very short text (such as hyphens and commas) between lists of hyper-links. However, it has been tested and perfected on many thousands of Web pages, directory listings and forms.

9 Security & Access Control

An important design goal is the provision of adequate security mechanisms. Existing mechanisms on the World Wide Web use per-directory access control [Berners-Lee et el., 1996; Franks et al., 1997]. Since we expect Satchel to be used with private, confidential documents, not necessarily separated from less sensitive ones, we have designed and implemented the secure per-document access control mechanism described below.

All tokens issued by a Satchel Browser are signed, using public key cryptography [see Salomaa, 1990]. Two keys, one public and one private, are used for this purpose. The private key resides in the mobile Browser, and never leaves it. The public key is available to anybody who wishes to use it. The two act as a pair, such that anything encrypted or signed using the private key can be decrypted or verified using only the public key (and vice versa).

Encryption is a very time consuming process, especially on mobile devices with limited computing power. Therefore, in the Satchel system, each token is boiled down to a short string, using a carefully chosen hash function, and just this ‘digest’ is encrypted to form the signature, rather than the whole token.

Given a token, anybody in possession of the appropriate public key can quickly verify that the corresponding private key must have been used to sign it. Thus, a token cannot easily be forged (since only the Browser has the private key), and is tamper-proof (since almost any change to its contents would invalidate the signature). Indeed, a

token grants access to just one document — not to a site, machine, directory, or any other document.

Signatures are carried in tokens as HTTP headers, ignored by public Web servers and proxies. However, the Satchel Personal Web Server (§6) does perform signature verification before serving any document. Requests made to the server must contain HTTP headers containing the digital signature and the identity of the signer. This identity must be one that has been registered in the server, typically that of the owner of the files it serves. The identity is used to obtain the appropriate public key from a public key registry, which is then cached, for future use. Should the signature check succeed, the HTTP request is honoured in the usual manner. Should it fail, either the cached key is no longer valid and must be refreshed, or an illegal request has been made. To determine which, the key registry is contacted once again: if the key is still the same, or if a second signature check fails with the new key, access is denied; otherwise the server's cached key is updated. In this manner, public keys automatically propagate from the registry to the Personal Servers throughout the system, and the vast majority of legitimate requests do not normally need to contact the registry.

In our prototype system, the public key registry is merely another Web-based CGI script, which returns a given user's public key from a simple database. The database itself must be protected from abuse, to prevent the falsification of public keys, and so is editable only by an administrator. Of course this is still vulnerable to more sophisticated attacks, such as spoofing the reply from the key server. In the future, we anticipate the use of trusted third parties, such as X500 Certificate Authorities [Weider and Reynolds], whose replies can be verified.

Loss or theft of the mobile Satchel device might suggest that the user's keys be changed. Although this is easily accomplished, doing so would invalidate previously distributed tokens, such as those Beamed or e-mailed. Alternatively, the private key in the mobile device might be relatively safe, since the Nokia 9000 can require a security code to be entered to use the PDA and phone. In any case, unwarranted access attempts can be monitored, since the Personal Web Server writes an access log.

Beamed tokens are also signed, which leads to a complication in their management. From the user's point of view, the Browser's Inbox page behaves like any other page, with links to documents. However, should one of these documents be fetched, or a document service be applied to it (including Beaming), the browser must use the originally beamed token verbatim in the request, since it was already signed elsewhere, and might refer to a private document. In contrast, normal links effectively contain a URL [Berners-Lee et al., 1994], and fresh tokens are generated and signed, dynamically.

In order to gain access to documents on an Intranet from the outside world, two strategies have been employed. The simplest was to install ISDN dial-in facilities, which can check the caller's phone number, thus verifying that a particular mobile device is in use, or denying access from unrecognised or barred numbers. Since the GSM network can connect directly to ISDN lines very quickly, this was our preferred route.

Another, more general method was to install a signature-checking proxy server on the 'firewall', between the Intranet and the Internet at large. This proxy examined signatures just as the Personal Web Server does, and allowed through legitimate requests to a nominated server. Thus, an external ISP could be used by the Browser to gain Internet access. In addition, tokens given to outsiders can be used as normal, provided the firewall proxy is completely transparent. Although technically straightforward, corporate security policies make this option politically difficult in some environments, especially since obscure bugs might leave the entire Intranet open to abuse.

10 Testing and Evaluation

The Satchel System has been through many tests and evaluations. One of the early Apple Newton prototypes was evaluated in two trials within our own lab; the first within our own research group and the second with other members of the lab. The results of each of these small-scale trials heavily influenced the design of the current Satchel system, primarily the user interface of the browser, but also architecturally. The Satchel system, using the Nokia 9000 Communicator and Microsoft Windows NT, was deployed and then evaluated in a large-scale, seven week trial within our own company [see Lamming et al., in press, for a more detailed report of this trial]. There were 26 people in the user group, all part of a European product-launch team; they were distributed across three sites in southern England. This trial is briefly described below.

10.1 Trial Activities

The deployment activities included: installing Satchel Server software on three PCs, one at each site; Satchel-enabling eight printers, four scanners and 24 PCs (including installing Windows NT, bar two users who did not wish to change) with infrared transceivers and Gateways; installing Personal Web Servers on those PCs; and installing the Satchel Browser on 26 Nokia 9000s. We provided a route for radio communications to the Xerox Intranet, initially through a secure proxy on our firewall, but later, through an ISDN remote access router at our own lab in Cambridge.

We trained the users at the start of the trial, and then provided technical support, on site for the first few days, then by phone, fax, and SMS. Feedback from the users was obtained by: regular fieldworker visits to trial sites; recording meetings; interviewing users; obtaining user ratings of the severity of reported problems; questionnaires completed at the end of the trial.

10.2 Results of Trial

During the entire period of the trial, logs were kept of the number of Print, View, Scan, Fax and Fetch service requests. Table 2 shows the number of requests for each of these services, the number and percentage of failures, and the reasons for failure. Beam Service requests were not logged, but by detailed examination of the logs and the contents of each user's Inbox, we know that at least 58 Beam requests occurred. (Note that the E-mail Service was added to the current Satchel system after this trial.)

The most common cause of failure across all service requests was that either a laptop running the Personal Web Server, or another host PC, was removed from the network, thus making the documents inaccessible. The next common cause of failure was that tokens could become "stale" because users changed the locations of documents once a token had been created (see §10.3). A third common cause of failures was caused by public keys becoming inaccessible due to an intermittent bug in the Personal Web Server. Conversion problems included dealing with compressed file formats, Microsoft Office template files, and differing US letter/A4 page sizes.

Many Print Service failures were due to Satchel's server account losing appropriate permissions or the password being changed without our knowledge, by local administration staff. For the View Service, some logged failures were due to the viewing application being left open on the viewing PC; however, they were not noticed by users. One other View service failure was caused by a bug in the browser software on the Nokia 9000 corrupting the file name, hence making the document inaccessible. The Scan Service had the highest overall failure rate, mainly due to scanner problems, for example the scanner being switched off or paper jams.

Despite the rather high failure rates, users rated the reliability of Satchel as somewhat above average, for a prototype, compared to other systems in use at their sites. The Browser suffered from some reliability problems, as memory management under GEOS is difficult, generating most of the negative comments, rather than the Satchel system as a whole.

Of the 2,232 requests shown in Table 2, 79% were made via infrared and 21% via radio. The high percentage of requests using infrared is perhaps due mainly to two factors: using infrared was quicker and cheaper; and GSM coverage at two of the sites was very patchy, making radio requests difficult in some locations.

10.3 User Feedback

A detailed discussion of the feedback from users can be found in Lamming et al. [in press]. Feedback from users indicates that: (1) accessing remote documents was easy and was highly valued. There were several occasions when users required critical documents, during meetings or while working at home. But even if documents were not actually required, some users commented that it improved their confidence of documents being available even if they were not actually required. (2) Access to document devices was similarly easy. The strongest evidence for this comes from the high usage of the Print Service while users were at their own sites: some preferred it to accessing printers from their own PCs. (3) Sending electronic documents to other people was also easy, although many users felt it should be even simpler, perhaps requiring only one button press rather than two.

During the course of the trial, there were a number of situations in which unanticipated documents were accessed. Most of these happened in conjunction with scheduled meetings, either during the meeting itself or in a chance encounter before or after a meeting. For example, on one occasion a user accessed and then beamed a report to four other people at a meeting because he had forgotten to take the paper document with him. In several other cases, a document was accessed and then printed in response to a request from another person. Another person, when working at home, accessed a document on his office PC and had it faxed to his home fax machine because he had forgotten to bring it with him. Another found Satchel useful when he arrived home from work

Table 2. Number of requests, failures and reasons for failure of services.

Reasons for Failures	**Print**	**View**	**Scan**	**Fax**	**Fetch**	***Total***
Host removed from network	9	11		3	6	*29*
Stale tokens	11	7		4	6	*28*
Public keys unavailable	7	16				*23*
Conversion problems	13	4		1		*18*
No permission to print	15					*15*
Scanner problems			10			*10*
Viewing application left open		9				*9*
User chose inappropriate directory			6			*6*
User entered no fax number				4		*4*
Corrupted file name		1				*1*
Total Number of Failures	*55*	*48*	*16*	*12*	*12*	*143*
Requests Made	241	258	43	42	1,648	*2,232*
Failure Proportion	*23%*	*19%*	*37%*	*29%*	*1%*	*6%*

and realized that he had forgotten to fax a document to someone in the US. Another trial user unexpectedly met a foreign colleague in the corridor while away from his usual office. His colleague asked for a copy of a document, and our trial user was able to print the document on a Satchel-enabled printer near where they met.

A few problems were highlighted by the trial. For example, Satchel did not provide access to the users' Microsoft Mail messages or attachments, which some people used as a means of long-term file storage. Scrolling through long directories was considered far too slow. Other issues included the weight of the Nokia 9000, the quality of GSM coverage (both beyond our control), insufficient options on Print Service forms, and limited feedback from the commercial fax server.

11 Future Directions

11.1 Emerging Standards

There are several emerging technologies and standards that may be of benefit to Satchel in the future.

Since we first implemented Halibut several years ago, specialised mark-up languages for use by mobile devices, such as Handheld Device Mark-up Language (HDML) [Unwired Planet, 1997] and now Wireless Mark-up Language (WML) [WML, 1998], have been developed. These specialised mark-up languages are more verbose than HTML (although they are designed to be compressed), and are intended to be human-authored. Like HTML, each tag has a corresponding closing tag, and named attributes to be checked. In our experience, the card and deck metaphor they employ is not particularly well suited for simple directory browsing, and automatic translation from HTML is not simple. In contrast, Halibut content is not expected to be directly authored. All content is authored in HTML and translated to Halibut on the fly.

The Wireless Application Protocol (WAP) [WAP, 1998], which incorporates WML, is an industry-wide attempt to provide an architecture and protocols for developing commercial applications and services for wireless mobile devices, such as pagers, PDAs and phones. The WAP specifications include a compression scheme, security, a mark-up language (WML) and a scripting language. Sadly, each is awkwardly different from existing Internet standards, despite bearing a close relationship to them [Khare, 1999]. However, WAP promises widespread use of a standard browser, on cheaper phones than the Nokia 9000, and it is possible that Satchel architecture could employ the WAP specifications in future implementations.

Wireless networking technology is constantly evolving and is a standard component in more and more devices — a clear benefit for Satchel. Bluetooth [Bluetooth, 1998], for example, promises relatively fast short-range radio communications, and does not require the degree of proximity or line-of-sight of IrDA [IrDA, 1997].

The industry is now beginning to exploit wireless data networks in ways that will integrate well with Satchel. One example is the BlackBerry mobile e-mail solution [Blackberry, 1999], which uses a forwarding agent running on a user's desktop PC to forward incoming e-mail via a wireless data network to a handheld device.

11.2 Room Service

We have experimented with alternative organisations of service directories, either by category (printing, scanning, etc.) or geography (here, this room, this floor, this building, etc.). We envisaged being able to search for the nearest device fulfilling some criteria, such as a colour printer with A3 capabilities, and a simple demonstration of this search capability was implemented in the Newton Satchel prototype.

Technologies such as Jini [Waldo, 1999] and Universal Plug and Play [Microsoft, 1999], offer lightweight impromptu access to services offered by a variety of digital devices, such as workstations, PDAs, mobile phones or home entertainment systems. A Satchel system that built upon these technologies would potentially allow the user streamlined access to a much wider set of devices and services.

11.3 Stale Tokens

One problem with tokens (and with references in general, e.g. URLs) is that if a file is moved or deleted, then any tokens previously handed out for it become useless. In the early Newton-based prototype, we experimented with an extra level of indirection to mask this effect. A user's directories only contained symbolic links to underlying files held in a special system directory. Pressing a SYNC button in the browser maintained this link-file mapping, and caused changes to be sent to the Newton. All tokens referred to the underlying file, rather than the symbolic link, thus freeing the user to rearrange or delete their 'files' at will. The major problem with this was knowing when an underlying file could actually be deleted, akin to garbage collection, so the Newton needed to keep a record of what tokens had been passed out. This data was used at SYNC time to determine conservatively which files could not possibly be accessed any longer (those for which the symbolic link had been deleted and no tokens had been handed out), and they were deleted. Unfortunately, the system rapidly became too cumbersome to manage and was abandoned. For example, a file once printed could not be deleted; a symbolic link moved out of sight of the system still worked, until the next SYNC mistakenly caused the underlying file to be deleted. However, a solution to this 'stale token' problem is still desirable.

11.4 Cache Consistency

Should a directory be updated whilst the user is on the move, the browser's cache may become inconsistent with reality. Since the browser does not hold cached copies of files, this is not quite the same as the file hoarding

problem addressed by [Tait et al., 1995], as the browser may well be able to hold a user's entire directory structure. If the user does become aware of inconsistencies, the browser's Reload button will refresh a cached page. At least the user's own file-space is unlikely to be modified with them out of the office, except perhaps by them scanning a document. An automatic directory reload after scanning, or after any service invocation might alleviate this particular case, but not the more general problem. While an explicit check of directory modification times could be made, this could be very slow.

A more sophisticated possibility would be to use a replicated database management system like Bayou [see Terry et al., 1998] to maintain consistency. However, since Satchel is essentially providing a simple view of a file system, unable to modifying it, the overhead of such a system would appear to be inappropriate.

In a later trial of the system, a program was developed that detected updates, additions or deletions to the document repository. When invoked, by hand, a textual SMS message was sent to each registered Satchel user notifying them of the change. Since the Nokia Satchel Browser can receive SMS messages itself, this simple idea could be extended to allow the Nokia Satchel Browser to automatically update directories on receipt of (specially formatted) SMS messages from a watchdog. Alternatively, an update file might be written, which could be checked upon next connection.

11.5 Enhanced Security

We envisage the ability to Beam tokens for commercially or politically sensitive documents destined for a known trusted recipient, via a distrusted third party, such as a courier or secretary. A token would need to contain the desired recipient's identity, and be signed as a whole by them when retrieved. Extra information could also be carried in the Beam Service form, which a recipient's browser transmits upon receipt of a service enquiry, to prove that the form was issued by the intended recipient, or somebody acting on their behalf.

Incorporating the date and time, or 'nonce' into the signed data of a token could produce tokens with a certain validity period [see Franks et al., 1997], either to reduce the impact of excessive widespread distribution, or to guard against replay attacks. More generally, an elaborated security model, perhaps incorporating a document access rights language, such as Digital Properties Rights Language [Ramanujapuram and Ram, 1998], might provide more fine-grained control.

Documents are not currently encrypted for transmission. However, SSL [Frier et al., 1996] or TLS [Dierks and Allen, 1999] could be used for these transmissions, at the cost of time and flexibility.

11.6 Document Identification

We envisaged one use of the View Service to identify documents in cases of doubt, perhaps prior to printing. However, users did not think this service useful, except perhaps with a projection screen for presentations. Alternative methods of identification through the Satchel Browser are under consideration, ranging from giving information about the file, or giving the first few lines of text-based documents, through to thumb-nailing the document's front page. Automatic summarisation is also a possibility. However, no one of these methods is ideal, especially in the case of selection amongst very similar documents, or differing versions of the same document.

11.7 Alternative Document Repositories

The current Satchel system accesses those files normally accessible from the user's PC desktop, including network-accessible machines. Closer integration with other document repositories, such as Xerox DocuShare or Lotus Notes would be useful. However, we know from trial feedback, that many users would like access to e-mail messages and attachments too.

12 Conclusion

Satchel has been proven, through user trials, to be successful at providing mobile access to documents and services. With a small device that professionals are generally willing to carry with them wherever they go, it provides **timely access to documents** through the use of a new simple concept, called *document tokens*. Tokens are small, and so can be transmitted quickly and cheaply over a wireless channel, while the documents to which they refer are still transmitted over the much faster wired network. Tokens are straightforward to create from small, easily cached directory listings. Even though wireless bandwidth will undoubtedly grow, so will the size of typical documents — while the pressure on the size and power consumption of portable devices will persist.

One reason for the success of Satchel is its **streamlined user interface**. Satchel is an example of an information appliance — we call it a *document appliance* — and as such its user interface is heavily specialised and thus very simple to learn, provoking few operator errors and requiring just a few button presses to perform an operation.

Satchel uses context to streamline **access to document services**. In front of a printer, Satchel presents a printing service, and in front of a scanner, a scanning service. This "Do What I Mean" interface significantly simplifies operations, belying the complex set of actions being performed behind the scenes, to fetch and then translate documents into the appropriate format for printing, for example.

Satchel is often present at the moment that the need for a document arises. The transaction can be initiated immediately and closure achieved; there is no need to remember to send a document on return to the office. However, there were occasions when GSM coverage was inadequate and there were no infrared transceivers nearby. Apart from Beaming tokens to other users, Satchel was quite useless in these situations. Fortunately, this was a

rare occurrence, but it nevertheless impaired our ideal goal of **ubiquity**. Similarly, the limited number of Satchel-enabled printers available in our trial was occasionally a problem, but this was substantially offset by the relative ubiquity of fax machines. By basing the Satchel architecture on the Web, we were able to deploy Satchel on a large scale with relatively little effort, and so we were able to provide almost world-wide coverage for the basic services of faxing and e-mailing. Indeed, Satchel has been used several times by the authors to print documents from our Cambridge, UK laboratory to a Xerox research center in Rochester, New York, and to scan documents in Palo Alto, California back to our Cambridge desktops.

A key feature of tokens is per-document access control. This **security mechanism** allowed the user to grant access to specific documents, without opening up the whole corpus of documents to the world. The token scheme is lightweight and transparent, and most trial users were totally unaware of its presence. Consequently, even documents stored behind a corporate firewall could be distributed, faxed and e-mailed without any of the usual complications, and so users felt little need to move documents to less secure, though more accessible locations.

The Satchel system was not just designed to fulfil technical goals, but users have been involved at every stage of development. The trial evaluation proved that, despite some problems, the current Satchel system is deployable, and provided strong support for our design goals.

Acknowledgements

We would like to thank Mike Kleyn for his work on the public key cryptography mechanisms used in this system. Also, our thanks go to Dik Bentley, Kevin Palfreyman, Mark Stringer and Chris Hines for reviewing drafts of this paper.

References

Terry, D. B., Petersen, K., Spreitzer, M. J., and Theimer M. M. December 1998. The Case for Non-transparent Replication: Examples from Bayou, In *IEEE Data Engineering*, , pages 12-20.

Baggio, A., and Piumarta, I. 1996. "Mobile host tracking and resource discovery", In *Proc. of the 7th ACM SIGOPS European Workshop*, Connemara, Ireland.

Berners-Lee, T., Fielding R. and Frystyk H. May 1996. Hypertext Transfer Protocol - HTTP/1.0. *RFC 1945*.

Berners-Lee, T., and Connolly, D. November 1995. Hypertext Mark-up Language - 2.0. *RFC 1866*.

Berners-Lee, T., Masinter, L., and McCahill M. December 1994. Uniform Resource Locators (URL), *RFC 1738*.

BlackBerry. 1999. Research In Motion introduces wireless e-mail solution for Microsoft Exchange users. *Press Release*. Research In Motion Limited, Waterloo, Ontario, Canada.

Bluetooth. 1998. Bluetooth technology overview.

Borenstein, N., and Freed, N. 1993. "MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies", RFC 1521.

Dierks, T. and Allen, C. January 1999. "The TLS Protocol", RFC 2246.

Donnelly C and Stallman R. 1995. *The Bison Manual: Using the YACC-compatible Parser Generator*, Free Software Foundation; ISBN: 1882114450.

Fox, A., Goldberg, I., Gribble, S. D., Lee, D. C., Polito, A., and Brewer, E. A. September 1998. Experience With Top Gun Wingman: A Proxy-Based Graphical Web Browser for the 3Com PalmPilot, In *Proceedings of Middleware '98*, Lake District, England.

Franks, J., Hallam-Baker, P., Hostetler, J., Leach, P., Luotonen, A., Sink, E., Stewart, L. January 1997. *An Extension to HTTP: Digest Access Authentication*. RFC 2069.

Frier, A., Karlton, P., and Kocher, P. November 1996. "The SSL 3.0 Protocol", Netscape Communications Corp.

IrDA 1997. Infra Red Data Association, Walnut Creek, California.

Khare, R. July/August 1999. W* Effect Considered Harmful, In IEEE Internet Computing, Vol. 3, No. 4, pp. 89-92.

Lamming, M. G. and Flynn, M. February 1994. "Forget-me-Not": Intimate Computing in Support of Human Memory. In *FRIEND21 Symposium on Next Generation Human Interface*, Tokyo, Japan.

Lamming, M., Eldridge M., Flynn, M., Jones, C, and Pendlebury, D. In Press. Satchel - Providing Access to Any Document, Any Time, Anywhere. Transactions on Computer-Human Interaction.

Microsoft. 1999. Universal Plug and Play Initiative Announcement. *Press Release.* Microsoft, Redmond, Washington.

NCSA Software Development Group. CGI: Common Gateway Interface.

McKeehan, J. and Rhodes, N. 1996. Programming for the Newton, AP Professional; ISBN: 012484832X.

Putz, S. 1993. Design and Implementation of the System 33 Document Service. Xerox Palo Alto Research Centre P93-00112.

Ramanujapuram A and Ram P. December 1998. *Digital Content & Intellectual Property Rights*, in Dr. Dobb's Journal.

Salomaa, A. 1990. Public-Key Cryptography. *Springer-Verlag Berlin Heidelberg.*

Tait, C., Lei, H., Acharya, S. and Chang, H. 1995. Intelligent File Hoarding for Mobile Computing . In *Proceedings of the First ACM Conference on Mobile Computing and Networking*, Mobicom 95, Berkeley. Page 119-125.

Unwired Planet. 1997. Handheld Device Markup Language Specification. Unwired Planet, Inc. Redwood City, California.

Waldo, J. 1999. Jini Architectural Overview. *Sun Microsystems*, Palo Alto, California.

Want, R., Schilit, W. N., Adams, N. I., Gold, R., Petersen, K., Goldberg, D., Ellis, J. R. and Weiser, M. December 1995. An overview of the PARCTAB ubiquitous computing environment, *IEEE Personal Communications* 2, 6 28-43.

WAP. April 1998. Wireless Application Protocol Architecture Specification. Wireless Application Protocol Forum Ltd.

Weider, C., and Reynolds, J. Executive Introduction to Directory Services Using the X.500 Protocol. FYI 13, RFC 1308, ANS, ISI.

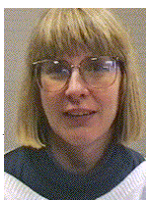
Weiser, M. September 1991. "The Computer for the Twenty-First Century", *Scientific American*, pp. 94-10.

WML. April 1998. Wireless Markup Language. Wireless Application Protocol Forum Ltd.

Mike Flynn received his degree in Computer Science from the University of Warwick in 1981. In 1982 he joined Logica Ltd, working in commercial communications and fault-tolerant systems then research into automatic VLSI design, formal methods, design tools and object-orientation. In 1991, he joined Xerox as a research scientist, producing the prototype Video Diary and Forget-Me-Not systems. He is the software architect of the Satchel project. His present research interest is speech access to documents and services.
E-mail: flynn@xrce.xerox.com

David Pendlebury joined the MDS group at XRCE Cambridge in 1994 as a Computer Science graduate of Cambridge University. He has contributed to the overall system architecture of several of the prototypes the group has developed. He has also been responsible for designing, prototyping and delivering several document services tailored for mobile users.
E-mail: david@aspectcapital.com

Chris Jones received his degree in engineering from Churchill College, Cambridge in 1994. He joined Xerox in 1995 as a research scientist to work on the Satchel project, specializing in the infrared and networking aspects of the system and implementing many of the prototype components. Since late 1998, he has been working for Xerox Mobile Solutions to develop Satchel into a product.
E-mail: cmjones@gbr.xerox.com



Marge Eldridge spent five years as the manager of a Human Factors and Graphics Design group at a Xerox engineering centre and then joined Xerox Research Centre Europe in 1990. Her background is in both experimental psychology and user interface design. Her present research interests involve studying mobile workers and identifying important problems in their mobile document work. She is also involved in the design of user interfaces for small, mobile devices.
E-mail: eldridge@xrce.xerox.com

Mik Lamming received his degree in computer science from London University in 1972. In 1982 he joined the Xerox Palo Alto Research Center (PARC) where he worked on digital printing and electronic photography. In 1988 he moved to Cambridge to help set up the European Research Centre where he leads a group pioneering interactive applications of wireless and mobile technology. He is co-author of the textbook Interactive System Design.
E-mail: lamming@xrce.xerox.com