m-Links: An Infrastructure for Very Small Internet Devices

Bill N. Schilit[1], Jonathan Trevor[1], David M. Hilbert[1], Tzu Khiau Koh[2]

[1] Fuji-Xerox Palo Alto Laboratory
3400 Hillview Avenue
Palo Alto, CA 94304 USA
+1 650 813 7220
{lastname}@pal.xerox.com

[2] Xerox Singapore Software Center
16 Science Park Drive #02-04
The Pasteur, Singapore 118227
kohtk@pal.xerox.com

ABSTRACT

In this paper we describe the Mobile Link (m-Links) infrastructure for utilizing existing World Wide Web content and services on wireless phones and other very small Internet terminals. Very small devices, typically with 3-20 lines of text, provide portability and other functionality while sacrificing usability as Internet terminals. In order to provide access on such limited hardware we propose a small device web navigation model that is more appropriate than the desktop computer's web browsing model. We introduce a middleware proxy, the Navigation Engine, to facilitate the navigation model by concisely displaying the Web's link (i.e., URL) structure. Because not all Web information is appropriately "linked," the Navigation Engine incorporates data-detectors to extract bits of useful information such as phone numbers and addresses. In order to maximize program-data composibility, multiple network-based services (similar to browser plug-ins) are keyed to a link's attributes such as its MIME type. We have built this system with an emphasis on user extensibility and we describe the design and implementation as well as a basic set of middleware services that we have found to be particularly important.

Keywords

Wireless, wireless web, web phones, middleware, proxy.

1. INTRODUCTION

The future Internet will include huge numbers of smart phones and other sub-palm-sized devices moving among wireless cells and accessing multi-media content. If current trends continue, we may see these devices outnumber traditional Internet terminals in the near future. Very small devices share common characteristics: small displays; limited input; lower bandwidth; slow processors; and small memories. Moreover, such devices continue to evolve further and further from the desktop computer platform on which current Web infrastructure is based.

It is generally accepted that new Internet terminals should be able to leverage the installed infrastructure of Web content and services. A number of research projects and commercial products have demonstrated ways to bring the desktop Web experience to mobile devices (see related work). The basic mechanism is *transducing,* or transforming content to take into account limitations in bandwidth, color-depth and screen real estate. It is our experience with one such transducing proxy, the Web Digestor, which motivated this current research.

ACM SIGMOBILE 7/01 Rome, Italy

Figure 1: Very small wireless devices, such as cell phones and PDAs, are increasingly used as Internet terminals. However, their extremely limited user interfaces makes leveraging existing Web content and services a challenge.

1.1 Experience With a Web Transducer

The Digestor [5] is an intelligent proxy that performs semantic compression and layout modification of web pages for a PDA or laptop. In other words, Digestor takes a web page and splits it into multiple web pages (each better suited for the smaller display) and adds new navigation links. The goal is to mimic the expert web designer if they were faced with the task of re-authoring web pages for PDAs. Digestor credibly transduced content for a range of small device types but broke down on very small devices. The problem is that transducing a desktop-sized UI into many pieces for display on a smart phone-sized UI inevitably results in a much more complicated structure that is difficult for users to understand and navigate.

Trying to use Digestor on smart phones led us to re-examine the desktop user's web experience and the desire to support that experience. We realized that much "browsing" involves following links and reading, or more generally, navigating to information and then using it. Moreover, we saw that activities performed on web content included reading but also mailing, printing, saving, and even translating. Such activities are well supported by the large UI of the desktop computer, but not by the limited UI of small devices. Returning our attention to small devices, we were motivated then not to transform the web, but rather to factor the web interaction.

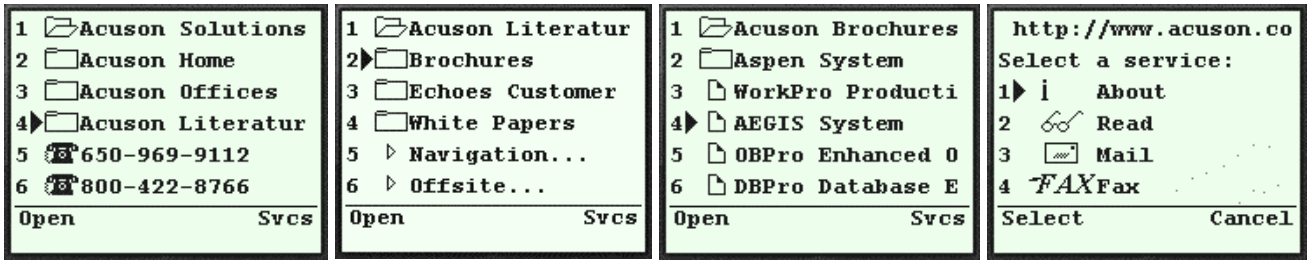

Figure 2: (a) Wireless micro-browsers hook into the m-Links network-side infrastructure through a URL that manufactures native format (CHTML, HTML, WML, HDML) screens. The m-Links Navigation Engine transforms the "fat" desktop web into a skeleton, more easily navigable form. Links, documents, mail addresses, and other useful bits of information such as phone numbers are returned. (b) The Navigation Engine makes it easy to dig into a site to uncover the content needed. (c) After digging to some spot, users can do useful things by invoking a client-side or server-side service on a link. (d) The service menu associates services to links based on link attributes such as MIME type.

By "factor" we mean to take the integrated activity of following links and reading (known as browsing) and divide it into two activities: *navigation* and *use*. By adopting this new web interaction model for small devices we both simplify the interaction and also extend the capabilities of what can be done with web content. For example, upon navigating to content it is now possible to do much more than just read. We will explain the benefits more fully through a scenario that describes the m-Links system in use.

1.2 Usage Scenario

Pino, a solutions consultant, is traveling to a customer meeting. In the taxi he hears a radio news story announcing a merger between his customer's main competitor and Acuson, Inc. Pino decides he must learn more about this company and also bring information to the meeting. He turns on his Internet phone but the stories at the Wireless Wall Street Journal are too short and too general.

If Pino were back at the office, he would simply go to Acuson's corporate Web site, dig around, read the news announcements and download and print out some of the Adobe Acrobat-format product brochures—or even email or fax these over to his customer. Instead, since he is in a taxi, he pulls up the m-Links site on his phone's micro-browser.

Since m-Links is a *site navigation engine*, Pino enters "acuson" and is shown www.acuson.com as a matching site. (Pino could have also used his history list or a search to arrive at the corporate Web site). As the taxi navigates the city streets of Rome, Pino uses m-Links' Navigation Engine to dig into the web site (see Figure 2a).

Pino sees the Acuson web site somewhat differently than it appears on his desktop computer. First, he doesn't see the content of the site, but rather a "skeleton view" showing the links. He also sees "data-detected" links representing phone numbers and addresses found on the page (Figure 2a). Selecting a phone number link would call that number.

Pino, however, is after product literature so is looking for that area of the web site. In Figure 2a he sees each HTML page link preceded by a folder icon (and as shown in Figure 2c each non-HTML file is preceded by a document icon). Since Pino is digging around the site this is just what he wants. He moves from the main page (Figure 2a) to the product Literature web page (Figure 2b) and follows that link to a web page with Brochures (Figure 2c).

Pino also interacts with the navigation engine differently than his desktop Web browser. Because most every small device is designed to select items from a list (e.g., phone numbers, contacts), the m-Links UI uses lists[1]. Therefore his interaction consists of scrolling the link list up and down, "opening" a link to its destination page or invoking an operation (a remote service) on a link, all of which can be accomplished by four buttons.

When Pino arrives at the brochures Web page (Figure 2c) he positions the cursor on the desired link item and calls up the services (Svcs) for that link. At this point a list of services appropriate for the MIME type of the link is presented (Figure 2d). These menu items connect to m-Links services (such as sending the link to another user) as well as existing Web-based services (such as language translation). Since Pino was interested in sending a fax or mail with the product literature file, he selects the Mail service and sends a message with the file as an attachment to his customer contact.

Although Pino often finds it difficult to read a desktop web page on his Internet phone, with m-Links he can comfortably navigate sites, especially when they follow a canonical layout, as do most corporate sites. Moreover, he can flip back and forth between the skeleton view and actually reading the content of the site (HTML, Adobe PDF, PowerPoint, and Word "readers" are link services).

1.3 Design Goals

The above scenario highlights some of the key aspects of the m-Links framework. Providing access to Web information and services on very small devices having only a few lines of text (or images) introduces numerous challenges. In our design we approach these challenges through a number of high-level goals:

[1] This design is reminiscent of both the early line-mode browsers, such as Lynx, as well as the familiar file selection dialogs.

Make Model	Network	Markup	Screen Size (HxW)	Dimensions (HxWxD)
Mitsubishi T250	CDPD 1.1	HDML WML	80x96 pixels 10x23 chars	200g 142x56x27mm
Mitsubishi D209i	TDMA	CHTML Color	96x90 pixels 8x7 chars	63g 125x40x15mm
NEC N209i	TDMA	CHTML Gray	108x82 pixels 9x6 chars	86g 90x46x19mm
NeoPoint NP1000	CDMA PCS	HDML WML	120x160 pixels 11x24 chars	181g 140x54x25mm
Palm Pilot VII		HTML Gray	160x160 pixels	190g 133x83x19mm
Qualcomm QCP-1960	CDMA	HDML	28x20 pixels 4x12 chars	120g 157x53x17mm
RIM 950	Mobitex	WML Gray	132x65 pixels	142g 63x89x23mm
Samsung SCH-3500	CDMA	HDML WML	96x32 pixels 4x12 chars	154g 112x52x25mm
Sony CMD-Z5	GSM	WML HTML	96x72 pixels 4x17 chars	82g 88x49x21mm

Figure 3: Characteristics of some very small wireless devices.

- **Web Navigation.** We wanted to make Web navigation on small devices faster and less disorienting, which led to culling the links from the content.
- **Get at useful bits of information**. We wanted a Web site's useful "bits of information," like phone numbers and addresses, to flow up to the user. This led to the server-side data detectors.
- **Maximize program/data composibility.** We wanted to emulate the desktop computer's ability to download content and perform many different operations on that content. The service menu with its list of services keyed by a link's MIME type allows this flexibility of operation.
- **Open Extensibility.** We wanted to be able to re-use existing web-based services as well as let users create their own services. Our mechanism for managing the underlying invocation and parameter passing to web-based services based on user profile information supports this.

1.4 Contributions

The main contribution of our work is the design and implementation of m-Links, a supporting infrastructure for Internet access over very small devices. This work departs from other Web transducing systems by proposing a different style of interaction, the navigation model, that we believe is more appropriate for very small devices than the desktop computer's browser model. Our design supports this navigation model and also breaks new ground by combining characteristics of search engines, desktop web browsers, and content transducers.

The remaining sections of this paper elaborate on the features highlighted in this scenario. The following section describes in more detail the navigation model underlying our approach. The next sections describe how m-Links fits into the existing wireless and Internet infrastructures, followed by a presentation of the components that make up the architecture, the server-side applications we have been experimenting with, and implementation details. The final sections present related work and conclusions.

2. A SMALL-DEVICE NAVIGATION MODEL

Today's "browser model" for accessing World Wide Web information evolved within the context of desktop computers with extensive user interfaces (displays, keyboards, pointing devices), considerable computing resources (CPU, storage, operating systems), and high bandwidth network connectivity. This model involves downloading and displaying HTML documents that include content (text, images, and user interface components) as well as links to other HTML and non-HTML documents (such as audio, video, Adobe PDF, and Microsoft Office files). When a user attempts to follow a link to a non-HTML document, the browser automatically invokes a client-side plug-in application. Such plug-in applications display the content and in some cases allow it to be manipulated and output using resources provided by the user's computer or other networked devices.

The success of the browser model is due, in large part, to the characteristics of networked desktop computers. Large displays allow rich content to be presented in conjunction with embedded links without sacrificing a user's ability to navigate the hyperlink structure. Full-sized keyboards and flexible pointing devices allow users to provide input to Web pages and plug-in applications without undue strain. Abundant CPU, storage, and operating system resources allow complex plug-ins to be executed locally in order to display, manipulate, and output Web content in various ways. Finally, high-bandwidth network connectivity allows media-rich content as well as sizeable plug-in applications to be quickly and easily downloaded to users' devices without compromising interactivity.

In contrast, today's small Internet terminals possess characteristics much different from the devices driving the browser application model. To illustrate these differences, consider the capabilities of some common small wireless devices (See Figure 3). The NeoPoint 1000, one of the larger Web phones in the U.S. market during 2000, has a screen capable of displaying 9 lines of 24 characters[2]. Like most web phones, it has a twelve-key numeric keypad that serves for both numeric as well as textual input. The NeoPoint also includes a small number of auxiliary keys to turn power on and off, start and end phone calls, select and activate features in the phone's display, and a 14.4 kbps wireless network connection.

While some of these characteristics are improving over time, especially in the area of higher-resolution color graphic displays, it is unlikely that they will change substantially due to the portability trade-offs.

Thus, instead of the browser model, we propose an alternative *navigation model* for accessing and using Web content on small devices. Whereas browsing involves an integrated activity of navigation and reading, the model we propose separates these into individual activities. From the user point of view, the m-Links navigation model embodies three steps:

[2] By way of comparison, DEC's (admittedly much more massive) VT100 terminal circa 1980 displayed 24 lines of 80 characters.

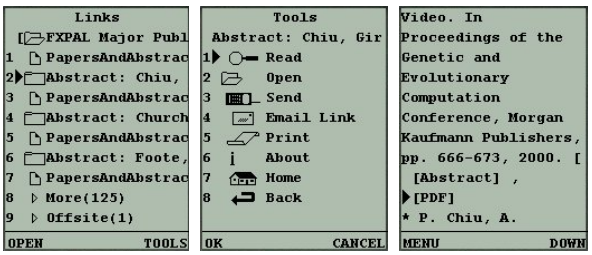

Figure 4: To make link labels more understandable m-Links tightly integrates a "reading" view with the link view so that users can expand and collapse the text surrounding a link.

1. The user requests a link (URL) to visit
2. The user is presented with a list of links and "digs" by repeating step 1 or decides to "do" something with the link destination content and goes to step 3.
3. The user is presented a list of services and upon selecting one, enters into that service with the target link as the primary parameter.

Informally we call this the "dig and do" model. Although the model appears simple, the realization raises design and implementation issues, especially in determining sensible labels for Web links, dealing with "link overload" from Web pages with huge numbers of links, handling information that is not directly linked but rather embedded in Web pages, and creating a high-degree of "open system design" in the services area.

Computing understandable and concise labels for links is a challenge when web page creators use anchor texts like "click here" liberally. Our design employs the notion of link label "quality" so that during processing the algorithm can compare various labels for the same link and select the best. Nevertheless, even with the quality metric (described in the next section), the issue remains that the context of a link informs the user. In other words, the text surrounding a link helps the user understand the content at the link destination. Clearly users will be confused upon seeing a list of phone number links without seeing their context in the original document.

To manage the basic problem of link context we have tightly integrated the "reading" service into the framework. This allows users to rapidly flip back and forth between the text of an HTML document and the links. A user can begin reading a page at the point where a link occurs. This gives the impression of expanding and collapsing the text around a link (see Figure 4).

Another difficulty faced by our model is "link overload" – or just having too many links to select from. (Actually, this is also a problem with the browser model having too large a page to display on a small device). Our basic approach is to provide automatic link categorization. Figure 2c shows categories for "Navigation" and "Offsite" which bring the user to lists of links associated with a navigation bar, and links that take the user offsite respectively. How these categories are detected is described later in the architecture discussion.

As you can tell, our model uses the link as a basic unit of manipulation, but what if the user wants to apply a service to non-linked information? To address this we introduced data-detectors within the infrastructure. This provides an elegant solution as long as a detector exists for the type of information users are interested in. We currently have detectors for phone numbers and addresses. Creating new links with the data-detected patterns reduces input demands on the users of devices with small and awkward input mechanisms. In some ways this approach works like cut and paste between applications on the desktop computer.

Finally, our proposed navigation model offers an opportunity for open system design that is as powerful as the browser model's use of plug-in applications. Whereas the desktop browsers associate a single viewer per MIME type, m-Links associates multiple services and lets the user choose among them. This also gives m-Links more of the feel of a desktop computer where multiple programs can be invoked on any given data file.

In sum, m-Links allows users to exploit a more Desktop-like application model that enables them to perform large device tasks on smaller devices.

3. DATA FLOW

Before presenting the m-Links architecture we describe how the system integrates into the existing wireless and Internet infrastructures.

The packet flow through m-Links is shown in Figure 5. Our system is designed to work with devices having an embedded microbrowser, such as cell phones and PDAs. Such microbrowsers are capable of accepting input from the user and displaying

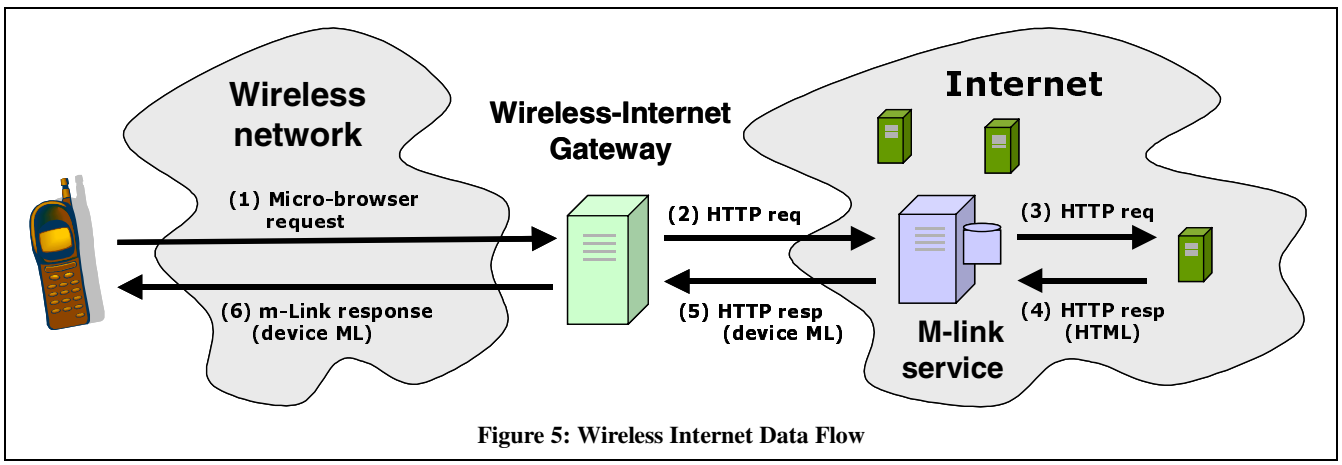

Figure 5: Wireless Internet Data Flow

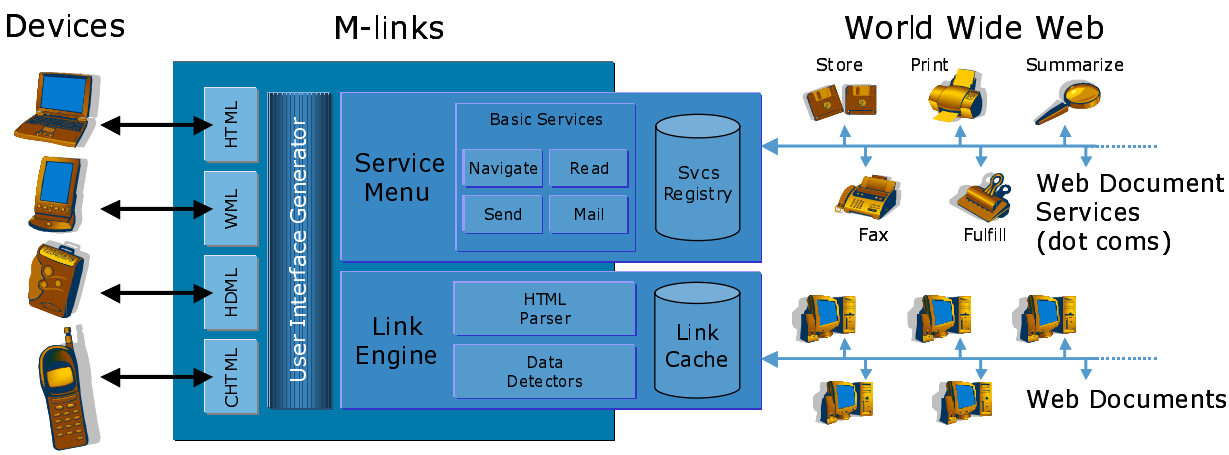

Figure 6: The m-Links architecture. Heterogeneous small wireless devices connect to m-Links using an embedded microbrowsers. The *User Interface Generator* converts all outgoing information into a form suitable for the device and browser. The *Link Engine* retrieves links from a Link Cache or fetches documents from Web servers and extracts link information. The *Service Manager* builds a list of appropriate services for a link based on MIME type and enables execution of web-based services.

information. There are a number of microbrowser available that employ various markup languages including HDML (Handheld Device Markup Language), WML (Wireless Markup Language), CHTML (Compact-HTML) or subsets of HTML. Our framework works with all these markup languages.

Microbrowsers communicate over an air-link network such as CDMA, CDPD, GSM, SMS or TDMA [10] to send requests to a wireless Internet gateway (1). A cellular telephone carrier such as Sprint, AT&T, Vodafone, or DoCoMo commonly operates the air interface and it is transparent beyond the gateway. The gateway unpacks the air-link data and forwards this information as HTTP requests to the Internet (2). The Wireless Gateway usually performs other functions such as acting as a cookie proxy. (Other configurations that include private Wireless Gateways are also possible).

When the m-Links server receives an HTTP request from the Wireless Gateway it uses HTTP header information to identify the microbrowser and device capabilities. Incoming requests are either satisfied locally or a Web server is consulted (3 and 4). In order to avoid this step, the m-Links server employs a large local store of Web page link information.

The m-Links server will respond to HTTP requests with link or service screens suitably formatted to device and browser characteristics in an appropriate markup language (5). The gateway then forwards this information to the air-link (6) where it is unpacked by the microbrowser and displayed.

This completes a single round-trip sequence between the microbrowser device and m-Links service. Generally this sequence occurs for each new page screen shown to the user, although to improve performance devices are incorporating screen caches and are able to pre-load screens.

In some ways the m-Links navigation engine is analogous to search engines such as AltaVista, Excite or Google. Users direct their (micro) browser to m-Links, enter a site name and get back a list of links. Users dig through the returned link information until a link to the desired content is found and then a transition is made away from the navigation engine to a service. Similarly, with search engines users direct their web browser to the search engine, enter keywords, get back a list of search results, refine their search, and jump off-site to view the content.

Another similarity between the navigation engine and a search engine is the use of crawlers. Our system includes a large store that holds the link structure of a part of the Web. We use a crawler to build up this database. Search engines also use crawlers to build up link information as well as a keyword index.

In other ways m-Links is more like a caching or a transducing proxy: when a link is requested that is not available in the database, the navigation engine goes out and fetches the page in real time and adds it's link information to the store.

4. M-LINKS ARCHITECTURE

4.1 Overview

The m-Links service has three main components (Figure 6). The Link Engine uses an HTML parser to extract links from web pages as well as label, categorize, and detect bits of information that should be converted into links. The Service Manager builds a service menu for a particular link and provides service hand-off. The User Interface Generator is the component that creates an appropriate user interface for a particular device and markup language. Each of these components is described more fully below, along with a discussion of scalability and methods for internationalization.

4.2 Link Engine

The link engine is responsible for processing web pages into a link collection data structure (see Figure 7). The Link Engine works with a Link Cache where link information for each processed page is stored. A request to process a web page involves these steps: (1) the document is loaded from the Internet using HTTP; (2) an HTML parser creates a parse tree; (3) the text elements in the parse tree are scanned by various data detectors for patterns (e.g.,

telephone numbers and addresses) and new links are created; (4) the links are categorized; (5) each link on the page is added to the page's link collection; and (6) the link collection data structure is stored in a cache.

4.2.1 Link extraction and naming

A basic part of the m-Links system is the extraction and naming of links from web pages. There are two types of link extracted from a given web page: explicit and data detected. Both are obtained by first passing the page to an HTML parser that creates a DOM (Document Object Model) for the page. Explicit links are those found in the HTML tags, such as anchors <A>, and image maps <AREA>. Data detected links are those which are present in the page but are not classified as links in HTML. Examples include physical street addresses and phone numbers. Each data detector receives the DOM and outputs these special links as they are identified.

Once the links have been identified the link engine employs a link naming algorithm to determine a concise and meaningful text label for the link. This is the label that is shown to the user. The algorithm identifies a variety of different possible labels for each link and assigns them a quality value representing how "good" or meaningful that link label is. The lowest quality label is the link URL itself. The highest quality label is assumed to be the title of the document at the links destination (for HTML pages) as page authors generally make titles meaningful for book marking. Other label sources falling between these extremes include: the anchor text of a link; the alt-text associated with an image link; and the link's URL path (excluding the host name and so on).

When different links (to different documents) share the same label the algorithm discards the label, moving to the next highest quality label, and re-checks the uniqueness of the new labels. This guarantees a distinct label for each different link appearing in the final user interface.

The link label quality metric allows a graceful degradation when poor labels are encountered. For example, a web site that titles each page the same would produce meaningless link labels if the only heuristic were to use the document title. With our technique we recognize these duplicates and use other labels until unambiguous labels are found. Additionally, the metric provides input to a back-off algorithm when it is too time consuming to examine all link destinations to determine the document titles.

```
<document ID="http://www.acuson.com/index.htm">
  <title>Welcome to Acuson</title>
  <http-headers>
    <last-modified>Wed, 15 Nov 2000 21:23:45 GMT</last-modified>
    <expires>none</expires>
    <content-type>text/html</content-type>
    <content-length>4552</content-length>
    …
  </http-headers>
  <links>
    <entry ID="/major_products.htm" offsite=0 navigation=1>
      <name quality=10>Major Products</name>
      <name quality=1>major_products </name>
      <name quality=0>www.acuson.com/major_products.htm
    </entry>
    <entry ID=http://www.siemens.com/indexEn.html"
      offsite=1 navigation=0>
      <name quality=4>Siemens Co., Ltd</name>
      <name quality=1>indexEn</name>
      <name quality=0>www.siemens.com/indexEn.html</name>
    </entry>
    …
  </links>
</document>
```

Figure 7: The extracted link structure represented as an XML document. Meta-information including HTTP header are stored along with all the links on the page. Each link has a variety of possible labels and categories.

4.2.2 Link Categorization

After links are labeled, the Link Engine categorizes them. An "off-site" category is assigned for links that refer to documents at a different web site from the document being processed. For example, when processing (1) below, links to (2) are considered on-site while links to (3) are not.

1. http://abc.here.com/index.htm
2. http://def.here.com/docs.htm
3. http://abc.there.com/main.htm

The categorization algorithm begins by extracting the server's domain name from the URL and discarding the protocol, port, and other components of the URL. To avoid mis-categorizing URLs that indicate multiple servers on the same site (e.g. 1 and 2 above), the algorithm constructs a "site identifier" for each URL by working backwards from the top-level domain (TLD) collecting up to two domain name components if it's a general, or gTLD (e.g., com, edu, gov, int, mil, net, org) and up to three components if it's a country code, or ccTLD (e.g., au, fr, uk, etc.). These site identifiers (e.g., "here.com" and "there.com") can then be compared against the site identifier for the page being processed ("here.com") to separate off-site from on-site links.

A "navigation" category is used to classify links that are used throughout a site to navigate from anywhere to common index pages. Navigation links are identified using a number of page layout heuristics. The categorization algorithm examines adjacent links and attempts to verify: (1) that they reside on the same hierarchical level in the parse tree, (2) that any intervening text between them is identical and acceptable (e.g., "|", "-", or "]["), and finally, (3) that the transitions, or intervening "paths" in the parse tree, between them are identical and acceptable (e.g., they occur in adjacent table cells or in the case of links with image anchors, may be separated by line-breaks
 or paragraph tags <P>).

Because the parse tree represents Web pages as containment hierarchies, the hierarchical level for each link can be determined by simply walking up the parse tree. If the hierarchy level is equal, then the intervening text and paths between navigation candidates is examined. Links with textual anchors (indicated by the <A> tag) or image anchors (indicated by the <IMG> tag) must occur in sequence with identical intervening text and paths between them. However, the paths that are acceptable between <A> links are not the same as the paths that are acceptable between <IMG> links. Finally, links occurring in image maps (indicated by the <AREA> tag) are assumed to provide site navigation functionality and therefore are not analyzed in the same way. We have found that while these heuristics are not foolproof, they are acceptable for the majority of the web sites we've examined.

The m-Links architecture is also capable of supporting link categorization based on MIME type (e.g., PDF files, MPEG files, MP3 files) as well as based on layout characteristics (e.g., links separated into frames, table rows, columns and cells).

4.2.3 Link cache

A cache services all outgoing requests from the link engine to the external World Wide Web. The cache is primarily seeded with all pages and documents from a large number of common web sites using a custom web crawler. When a request is made for a page that is not yet cached, it is fetched and added into the cache before being returned. This allows the cache to grow according to use. Subsequent requests for the same page will then result in a cache hit.

The cache behaves in a similar manner to those used by search engines; periodically updating existing cached pages as their original source changes externally. However it differs by storing the HTML page's link collection data structure and headers. By storing headers even for non-HTML documents, the Service Manager is able to rapidly generate the service menu.

4.3 Service Manager

The Service Manager is responsible for returning the subset of services that are appropriate for a link and user. There are two sources of services available to be the m-Links architecture: *general* and *content provider*. General services are web-based services hosted on particular sites that have previously been identified to the system. Users can specify which services they would like to see for which kinds of links.

Content provider services allow web site owners to control the services available for links to their web site. These are specified in a "services.xml" file at the root directory relative to the link[3]. This allows a content provider to include a set of customized services for their content. For example the web site www.patents.com might provide a service for overnight delivery of high quality patent documents.

The Service Manager accepts a link and generates an applicable set of services. First it checks for content provider services and then general services. In both cases, the generator evaluates a set of rules from the service specification against attributes of the link (e.g. the MIME-type of the link), the characteristics of the user's client device (e.g. what markup can the device display), and the user's identity (their email address). If all the service rules are satisfied then the service is added to the link's service menu. In this way a service developed to play audio files will appear only when an audio file is the selected link.

When one of these services is eventually invoked by the user, the Service Manager submits a HTTP request to the appropriate web server (determined by the service description – see below), identifying the target, the user, a return URL once the operation completes, and a number of optional parameters.

4.3.1 Defining and Extending Services

The service specification document is the mechanism by which both general and provider specific services are described. Figure 8 shows an abbreviated version of the XML-based specification for an email service.

There are three main sections in a service specification: rule; execution and presentation. The *rule section* describes when the service should be presented to the user as a possible service to be invoked. The section includes predicates that declare the required attributes of the link, the type of client device, and the identity of the user.

```
<service-group ID="email-group">
  <service ID="email">
    <rule>
      <accept match="any">
       <client-accepts>.*text/html.*
       </client-accepts>
       <client-accepts>.*text/x-hdml.*
       </client-accepts>
     </accept>
     <reject>
       <source-mimetype>URL/.*
       </source-mimetype>
      </reject>
    </rule>
    <execution>
        <execute>/mailto</execute>
    </execution >
    <presentation>
      <language ID="en">
       <service-name>email document
       </service-name>
       <short-name>Email Link</short-name>
       <description>
          Email a URL or the URL
          and its contents to yourself
          or a friend.
       </description>
     </language>
     <icon output-format="text/x-hdml">
          envelope1
      </icon>
    </presentation>
  …
  </service>
…
</service-group>
```

Figure 8: A Service Specification describes a service and determines which services are applicable for a link. The email service (above) includes an execution section describing how to execute the service and various pieces of presentation information for displaying the service across different types of menus.

The *execution section* describes what URL should be executed when the service is selected. The service is activated using an HTTP request with a number of parameters, including the link to operate on, the user identify and so on.

The *presentation section* is separated into subsections for different languages. Each language element provides short and long labels to present to the user, as well as a longer description. A set of icon tags provides links to various graphical elements that can also be used when displaying the service to the user.

The XML format for service specification allows one or more files to be included, and allows other external specifications to be referenced using URLs which are resolved and substituted during the validation of the XML.

Adding new services to m-Links system is as simple as submitting a URL with the location of the service description file. The m-Links system then fetches and checks the description for validity,

[3] In a similar way to the "robots.txt" file that controls how a web site should be crawled or indexed.

and adds it to the service registry. The next time a service menu is generated for a link the service rules will be checked and the service can appear in the menu.

4.4 User Interface Generator

m-Links supports a variety of different user-interfaces to handle the variety of devices that may access the system. For example, HDML and WML markup is used by web-phones, and HTML is used by some palm-size PDAs. However, while the markup supported by the various client browsers on these devices differs, the actual underlying functional and interaction model of the interface is very similar. For example, all the different markup language interfaces provide a screen where the user can input a web site to be navigated. The UI Generator exploits this shared functionality using a combination of "template markup files" and program inheritance. Together these support a multi-view interface for the different device types.

```
<html>
<head>
<title>Input</title>
</head>
<body bgcolor="#FFFFEE">
<h2><img border="0" src="mlinks.gif"></h2>
<table border="0" width="530" height="100">
  <tr>
    <td width="523" height="51">
      <h2>Web Links</h2>
    </td>
  </tr>
  <tr>
    <td width="523" height="40"><b>
    Enter the name or URL of the web site where you
    would like to visit :</b></td>
  </tr>
</table>
<form method="get" action=%(next)>
<table border="0" width="530" height="50">
  <tr>
    <td width="465" height="11">
      <input name="u" size="65" value=%(defaulturl)></td>
    <td width="465" height="11">
      <input type="submit" value="Go">  </td>
  </tr>
  <tr>
    <td width="465" height="27"><i>
     (E.g. cnn or bbc.co.uk)</i>
  </td>
  …
</body>
</html>
```

Figure 9: An HTML template markup file for the site address input screen

```
<HDML VERSION=3.0 MARKABLE=TRUE>
<ENTRY DEFAULT="%(defaulturl)" KEY=u FORMAT=*x>
<ACTION METHOD=GET TYPE=ACCEPT LABEL="GO"
       TASK=GOSUB FRIEND=TRUE DEST="%(next)?u=$u">
<LINE>Enter a web site:<LINE>Ex: cnn or bbc.co.uk
</ENTRY>
</HDML>
```

Figure 10: An HDML template markup file for the site address input screen.

When a request arrives to m-Links, the interface generator performs these steps: (1) identifies the type of device making the request; (2) determines the appropriate type of response markup; (3) extracts various pieces of information from the request (such as the HTTP headers); (4) dispatches it to the relevant interface markup handler (for the identified markup type) to generate the interface; (5) returns the markup to the device.

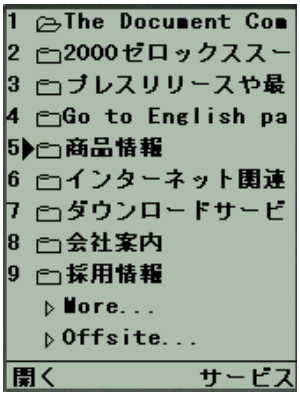

Figure 11: Internationalization involves passing language information from the original web pages to the link menu screen. Each screen template used by the UI generator includes version for different language as well.

Although the final markup may be very different from device to device, the actual variables involved remain the same. Therefore m-Links employs a simple template substitution technique across most of the different interface screens. The same base interface-code is responsible for generating the variable values from request to request for that interface. These values are substituted into named fields that have been inserted into the template for a given screen. Figure 9 and Figure 10 show the same web-site input screen for both HDML (for web-phones) and HTML (for desktop browsers). Note that while functionally identical both templates when completed would look very different, and the same named variables, "*defaulturl*" and "*next*" appear in both templates.

In more complex interface situations, where the supported functionality or methods of interaction differ significantly between different devices and markup languages, the base interface generator is sub-classed by the markup handler to extend and tailor the functionality**.**

4.5 Internationalization

The Web contains information in many languages. The m-Links system manages multiple languages in two ways. First, the font encoding for the target web page is passed through the Link Engine allowing the user to see link labels in the font intended by the author. Second, m-Links allows the language for menus, prompts, and messages to be specified by the user. This is facilitated by the use of template markup files for each language.

5. SERVICES

We have experimented with a number of services that are described below in broad categories.

5.1 Reading

Although reading of link content, such as MS Word or HTML, is difficult on small devices, we developed a number of reading (or preview) services for different file types because people wanted to "check the contents" before proceeding with another operation such as faxing. Each reading service extracts content from a type of file and presents it in a device-specific manner. For example, our PowerPoint reader extracts the text from slides rather than the graphics because our current set of devices are not graphic capable. By convention, the text from each reader service is presented in a simple linear format. In an earlier system [5] we experimented with different presentations, including summarizations based on the document structure (e.g., expandable

outline views using headings and sub-headings), but found the linear presentation less disorienting to users of very small devices. Although we had intended each reading service to be a completely independent component of the system, the HTML reader evolved into a tightly coupled component because it was useful to flip back and forth between the link and the reading views. This was desirable, for example, to see the document context of a telephone number and other non-intuitively named links.

5.2 Sending

We developed both email and alert style link communication as services. The email service can send the link, and optionally the link contents as an attachment, to one or more email recipients. The email message itself contains a URL pointing at the m-Links service menu page for the link. Therefore the email recipient can immediately enter the m-Links system (on the desktop or their small device) and be presented with a set of appropriate services to act on that link.

The WAP-alert service exploits the alert functionality of the UP.LINK gateway used by some WAP phones. When the alert service is invoked on a link a buddy list of users with WAP phones is presented. Selection causes a WAP-alert to be sent to that user's phone, which will popup a message on the phone. If the recipient accepts the message their WAP mini-browser takes them to the m-Links page for that link – allowing easy access to useful tools for the link.

5.3 Printing

We developed a number of print and fax services to deliver hard copies of the contents of links to users. Our fax service was developed as a wrapper around an existing web-based service (www.eFax.com). The wrapper accepts the target link and prompts the user for a fax number. It then downloads the document and sends it as an email attachment to the fax service. The print service uses a print-queue monitor application that runs on a user's networked desktop computer. When the print service is invoked, a link is added to the user's print queue. The monitor detects the link, downloads the contents to the user's local disk and invokes the appropriate application to print.

5.4 Mapping

We developed two services that operate on data-detected address links. They both employ the Yahoo on-line mapping service to obtain data. The first provides text-only directions to the address. The second displays a list of various maps at different zoom levels. These maps are links to GIF images on the web. When a map is selected the service invokes the m-Links system again to display possible service operations for the particular (GIF) map, allowing the user to fax, print, or email the map.

6. IMPLEMENTATION & EXPERIENCE

Our implementation of the m-Links system is based on Java servlet engine running under Microsoft's IIS web server. We are running our system on a Pentium III processor with 256 MB of memory connected to a T1 network connection. Most of our early work employed a DSL connection that was suitable for all but massive crawling activities.

We seeded the current system by web crawling 5,000 sites selected at random from the Yahoo directory. This produced a few hundred megabytes of extracted link structure for the cache. We also produced a Yahoo style categorized directory listing pointing at these sites as a test "bookmark" list.

One issue we have found with the implementation is that some web pages contain client-side scripts (such as Javascript or VBScript) that are executed by the user's web browser. This allows features such as dynamic menus to be constructed based on the user's interaction with the page. These types of pages are a problem for m-Links as well as search engines since it is impossible to evaluate such scripts out of context and away from the user. In practice this problem is not so severe for two reasons. First, authors of many scripted pages also provide "hidden" or extra links for non-script supporting browsers to display. These are picked up by our HTML DOM parser and fed directly into the link engine. Secondly, many sites are aware of this problem but also want indexing by search engines and therefore provide alternative pages which link into the site contents.

7. RELATED AND FUTURE WORK

The m-Links system is related to middleware services that support wireless devices [1][2][3][5][7][8][15][17] as well as the general approach of using transducers on World Wide Web content [4]. One way we categorize these mobile web systems is by the target device size and capabilities.

The Web Digestor was developed to run on a range of devices but was most effective on PDAs. It employs a number of re-authoring techniques including outline summarization and elision of all but the first sentence of paragraphs. As mentioned earlier, Digestor (as well as other systems) followed the tradition of the desktop-based browser by presenting both content and links together. For the relatively large PDA devices, other elegant integrated solutions are possible. For example, the PowerBrowser presents both content and links using a technique in the tradition of fisheye views where a large body of information is displayed in progressively greater detail, with surrounding context always visible to some extent [6]. Other examples are Microsoft's Pocket and Mobile Explorers that faithfully reduce web pages on devices with high resolution displays using a shrink to fit feature [12].

However, as Internet devices get smaller these techniques become increasingly problematic. One approach used by Mobile Yahoo for phone-sized devices is to customize the presentation from the server using an existing, manually authored link database. Similarly, Xdrive [16] and Satchel [11] provide easy interfaces to access files from certain remote file stores and perform (limited) document-specific operations. Mobile Google generalizes by allowing users to search for and display World Wide Web HTML content on their mobile phones. The m-Links system goes further by allowing any World Wide Web content to be accessed (not just HTML) and provides an open architecture for adding services (not limited to reading).

The link-centric interface of m-Links exploits the notion of Data Detectors introduced by Apple so that non-linked data such as addresses can be associated with services [13].

We have been using the system over the last 6 months in the U.S. on a wide variety of Sprint Web phones and in Japan on DoCoMo CHTML web phones. Although the m-Link implementation has difficulty producing manageable results for pages with hundreds of links, we feel the navigation model is still the most promising approach for leveraging Web content on very small devices.

In the future our work will focus on: expanding the categorization capabilities to handle link-heavy pages; integrating a search engine to help find a starting point for navigation; and providing a generic mechanism for passing data from the user or user profile to services.

8. CONCLUSIONS

Our experience trying to use one of the early Web content transducers led us to reconsider the goal held by many to browse the Web. Instead we recognize that browsing requires a large interface not available on small devices and furthermore that reading content is difficult at best on small devices.

We propose a new interaction model called the *navigation model* for very small Internet terminals that has fewer requirements on the UI than the desktop computer's browsing model. We have presented an infrastructure design and described an implementation of our m-Links system supporting this model.

The m-Links system addresses our design goals: (1) supporting web navigation on very small devices; (2) getting at useful bits of embedded information on web pages; (3) maximizing program-data composibility through a separation of service from link; (4) providing a open framework for others to develop new services for wireless devices.

9. ACKNOWLEDGMENTS

We thank Andreas Girgensohn who helped develop the Digestor system along with Debra Go, Patrick Ahern, Eric Hope, Mik Lamming, Elizabeth Churchill, Cathy Marshall and Harry Saddler for their useful feedback. Finally, we thank Jim Baker and FX Palo Alto Laboratory for supporting this research.

10. REFERENCES

[1] Brewer, E., Katz, R.H. et. al. A Network Architecture for Heterogeneous Mobile Computing. IEEE Personal Communications Magazine (October 1998).

[2] Bartlett, J.F. Experience with a Wireless World Wide Web Client in Proceedings of IEEE COMPCON 95 (San Francisco, March 1995).

[3] Beck, J., Gefflaut, A. and Islam, N. MOCA: a service framework for mobile computing devices in Proceedings of the ACM International Workshop on Data Engineering for Wireless and Mobile Access (Seattle WA, August 20, 1999) 62-68.

[4] Brooks, C., Mazer, M., Meeks, S., and Miller, J. Application-Specific Proxy Servers as HTTP Stream Transducers in Proceedings of the Fourth International World Wide Web Conference (Boston MA, December 1995).

[5] Bickmore, T. and Schilit, B.N. Digestor: Device-Independent Access to the World Wide Web in Proceedings of the Sixth International World Wide Web Conference (Santa Clara CA, 1997).

[6] Buyukkokten, O., Garcia-Molina, H., Paepcke, A., and Winograd, T. Power Browser: Efficient Web Browsing for PDAs in Proceedings of the Human-Computer Interaction Conference 2000 (CHI 2000) (The Hague, The Netherlands, April 1-6, 2000).

[7] Fox, A. and Brewer, E. Reducing WWW Latency and Bandwidth Requirements via Real-Time Distillation, in Proceedings of the Fifth International World Wide Web Conference, World Wide Web Consortium (Paris, France, 1996)

[8] Fox, A., Goldberg, I., Gribble, S.D., Lee, D.C., Polito, A., and Brewer, E.A. Experience With Top Gun Wingman, A Proxy-Based Graphical Web Browser for the USR PalmPilot in Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware '98) (Lake District, UK, Sept. 1998)

[9] Gessler, S., and Kotulla, A. PDAs as Mobile WWW Browsers in Proceedings of the Second International World Wide Web Conference (Chicago, October 1994).

[10] Harvey, F. The Internet in Your Hands, Scientific American, 283(4) (Oct. 2000).

[11] Lamming, M., Eldridge, M., Flynn, M., Jones, C., and Pendlebury, D. Satchel: providing access to any document, any time, anywhere. ACM Transactions on Computer-Human Interaction, 7(3) (2000).

[12] Microsoft Corporation. Pocket Internet Explorer. http://www.microsoft.com/mobile/.

[13] Nardi, B.A., Miller, J.R., and Wright, D.J., Collaborative, Programmable Intelligent Agents. Communications of the ACM 41(3) (March 1998).

[14] Pandit, M.S., and Kalbag, S. The selection recognition agent: Instant access to relevant information and operations in Proceedings of Intelligent User Interfaces. (New York, 1997), ACM Press.

[15] Schilit, B.N., Douglis, F., Kristol, D.M., Krzyzanowski, P., Sienicki, J., and Trotter, J.A.: TeleWeb: Loosely Connected Access to the World Wide Web in Proceedings of the Fifth World Wide Web Conference (WWW5) (Paris, France, 1996) in Computer Networks 28(7-11) 1431-1444

[16] Xdrive Technologies. An Online Storage Box for Everyone. http://www.xdrive.com

[17] Zenel, B., and Duchamp, D. A General Purpose Proxy Filtering Mechanism Applied to the Mobile Environment in Proceedings of MOBICOM (1997), ACM Press, 248-259