Using Wrappers for Device Independent Web Access: Opportunities, Challenges and Limitations

Extended Abstract

Juliana Freire

Bell Laboratories

600 Mountain Ave.

Murray Hill, NJ 07974, USA

{juliana}@research.bell-labs.com

Abstract

The availability of technologies that enable mobile access to data has brought great expectations that users would be able to access information, entertainment and e-commerce any time, anywhere. However, the existing Web infrastructure and content were designed for desktop computers and are not well-suited for other types of accesses, e.g., devices that have less processing power and memory, small screens, and limited input facilities, or through wireless data networks with low bandwidth and high latency. Thus, there is a growing need for techniques that provide alternative means to access Web content and services, be it the ability to browse the Web through a wireless PDA or smart phone, or hands-free access through voice interfaces. In this paper, we discuss issues involved in providing ubiquitous access to Web data. We present techniques and systems for building wrappers that address these issues, and discuss their features and limitations in different application scenarios.

1 Introduction

The ability to access the Web *anytime, anywhere, and from any device* is a fascinating prospect. However, the existing Web infrastructure and content were designed for desktop computers and are not well-suited for other types of accesses, *e.g.,* devices that have less processing power and memory, small screens, and limited input facilities, or through wireless data networks with low bandwidth and high latency. Thus, there is a growing need for techniques that provide alternative means to access Web content and services, be it the ability to browse the Web through a wireless PDA or smart phone, or hands-free access through voice interfaces.

Example 1.1 (Pricing Airfare)   Consider the following scenario. Juliana plans to attend the WWW2002 conference and she is looking for flights from Newark Airport to Honolulu that leave from Newark on May 3rd and return on May 12th. From her desktop, she goes to `www.travelocity.com` and in a couple of minutes, after navigating through 4 pages, and having 400 Kb of data transferred, a page with the list of nine flights (as well as ads and additional navigational information) is displayed. Now, assume Juliana is away from the office and trying to access Travelocity using her Palm Pilot connected to the Internet through a wireless modem (whose effective throughput rates that vary from 5-6 kbps up to 12-13 kbps). She starts the Palm Web Clipper version 3.2 and inputs the Travelocity URL. It takes a few minutes just to retrieve the initial page, and to view the full page on the 160x160 pixel, she must scroll down 16 times. Besides the

	Phone	Palm Pilot	Typical Laptop
screen size	N/A	160x160 (6x6cm)	1024x768 (14")
bandwidth	N/A	5-19kbps	56kbs
input/output	keypad and voice/voice	Grafitti/subset of HTML	keyboard/HTML,XML,voice
processor	N/A	16-20Mhz	1GHz
memory	N/A	8MB	500MB

Figure 1: Device Capabilities

high latencies inherent to wireless networks, additional delays are incurred for transcoding the content into a format that can be displayed on the Palm browser. Since transcoding is a rather complex task and very error-prone, it is not surprising that Juliana was not able to access the flight list, in fact, she was not able to go beyond the login page: the submit button was not active and could not be clicked on.
□

Whereas there have been many promises and expectations for the ubiquitous Web, the dream has yet to materialize. The Web is just too complex and does not really scale well to the wide variety of Web-enabled devices that have widely different screen sizes, network connections, input and output, processors, etc. (See Figure 1.)

Web developers are faced with numerous decisions to make content available to diverse devices. There are different approaches and each of them has benefits and drawbacks depending on application requirements. In terms of user experience, the best possible strategy is to *design device-specific interfaces*. Some content providers have adopted this approach, for example: The New York Times has a palm-friendly section [14]; Amazon provides a specialized interface for Web-enabled phones, as well as for the Palm VII [1]; and various other Web sites now have mobile phone-friendly versions (see [17] for a list such sites). However, from a content-provider's point of view, creating maintaining multiple versions of an application can be expensive. Systems and standards have been proposed which try to mitigate the maintenance costs. SISL (Several Interfaces, Single Logic) [4] is an architecture and domain-specific language for structuring services with multiple user interfaces. By separating the logic and presentation aspects of an application, SISL creates a single specification for an application from which multiple interfaces can be automatically derived that are suitable for different devices. The W3C CC/PP (Composite Capability/Preference Profiles) work-group is developing standards for content negotiation that include profiles that describe users' preferences and device capabilities, content annotations and rules for adapting content to the capabilities and preferences of the user. Even though these proposals may lead to a cost-effective solution for designing new sites, adapting existing sites can be prohibitively expensive, as many changes may be required or even a complete re-design.

Transcoding proxies and wrappers offer an alternative to export different views of a Web page or service that does not require changes to the Web sites. *Transcoding proxies* [15, 8, 9] perform on-the-fly content adaptation, and in theory, they are a good general solution for allowing users to browse any Web site from any device. For example, since browsers on the Palm do not fully support HTML and are not able to display GIF or JPEG images, a Palm proxy must translate pages into the supported subset of HTML and images into bitmaps supported by the browser. In practice, however, proxies have significant limitations. Since these transcoders must handle any content and present Web pages as faithfully as possible, they do not perform any personalization and may generate content that is not well suited for a particular device. Besides, some features present in HTML pages are hard or even impossible to translate (*e.g.,* client-side script) and it is not unusual that proxies fail to properly transcode complex pages, or even simple, but badly designed pages.

Specialized *Web wrappers*, on the other hand, can be used to to create different views of Web sites and services that, unlike general proxies, can be both personalized and customized to specific devices [12, 11]. Web wrappers perform two important tasks: access – retrieve a particular page (or set of pages); and extraction – extract specific elements (or clippings) from the retrieved pages. Although the Web was designed for human interaction, by automating navigation and performing customized extraction, wrappers give applications control over the retrieval of pages and allow them to more effectively process Web content. Consider the scenario described in Example 1.1. A wrapper can be created that provides a *shortcut* to the flight list page and extracts from that page *just* the list of flights – in effect, giving the user a one-click access to the flight list. In fact, as we discuss below, wrappers can be used to export different parameterizable *views* of Web sites that are better suited to be accessed from different terminals.
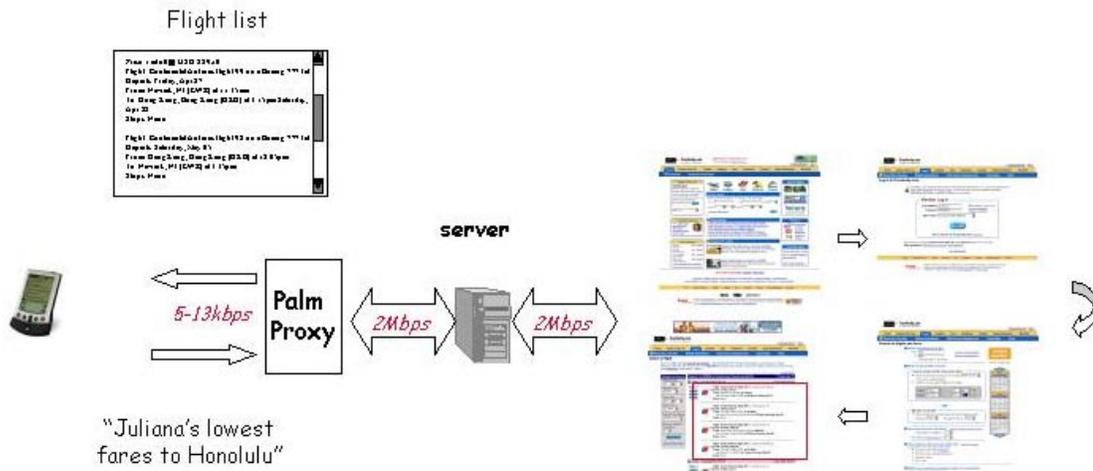

Figure 2: Accessing Web content through customized wrappers

The benefits of Web wrappers go beyond simplified interaction. As Figure 2 illustrates, the wrapper can executed at a server (with a fast connection to the Internet) greatly improving response times: ignoring latencies, downloading the 400kb required to access the flight list from Travelocity takes anywhere between 240 and 600 secs over CDPD [1], whereas from desktop computer connected to the Internet through a cable modem the transfer would take less than 5 seconds. In addition, transcoding is greatly simplified: instead of transcoding each page, only subset of the final page needs transcoding.

Wrappers thus offer an attractive alternative to deliver Web content to diverse devices. There are however many challenges involved in building an effective wrapper-based solution, most notably: scalability and robustness. If many different wrappers need to be created, the process to create and maintain these must be *simple*, and this is specially hard to attain as Web sites become increasingly complex. In addition, wrappers should be robust to common changes to Web sites, or at least degrade gracefully as sites evolve.

In this paper we give an overview of Web wrappers and how they can be used to provide ubiquitous access to Web content. We discuss different approaches to wrapper creation, their features and limitations. Note that it is not our intent to provide an exhaustive survey of the area, instead we focus on key issues and solutions. The structure of the paper is as follows. We start in Section 2 we give an overview of wrappers

---

[1]CDPD [6] is a wireless IP network that overlays on the existing AMPS (analog) cellular infrastructure.

```
import Str;

var P = PostURL("http://people.yahoo.com/py/psPhoneSearch.py?Pyt=Tps&YY=2898",
                    [. FirstName="John", LastName="Smith".]);

PrintLn(P);
```

Figure 3: WebL program to looks up e-mail addresses Yahoo!

and the main issues involved in creating wrappers. In Section 3 we describe a wrapper-based architecture for content delivery and some important features that should be supported by a wrapper API. The interaction between wrappers and transcoding is discussed in Section 4. We conclude with a discussion in Section 5.

2 Web Wrappers

Web wrappers automate the retrieval of Web pages and the extraction of components or information from retrieved pages – they encapsulate the actions required to retrieve a particular page, along with the specification of which elements should be extracted from the retrieved page. Given the growing trend of creating interactive and dynamic Web sites that publish data on demand, retrieving information from the Web is becoming increasingly complicated for humans, and even more so for wrappers. In what follows, we give an overview of techniques to create access and extraction wrappers.

2.1 Automating Navigation

The task of an access wrapper can be as simple as issuing an HTTP GET to retrieve a document. However, Web sites have become quite complex. From online classified ads to banks, many sites require users to fill out a sequence of forms and/or follow a sequence of links to access a particular page. Often, these pages do not have a well-defined URL. For example, below is the URL of the flight list in Travelocity:

```
http://dps1.travelocity.com/airgchoice.ctl?
         SEQ=10191535725604270418002&LANG=EN
```

If an HTTP GET is issued for this URL, a page is returned with an error message. Thus, in order to retrieve *hard-to-reach* pages such as the flight list in Travelocity, wrappers must mimic a user's actions, and execute a sequence of navigation steps. There are different alternatives to create access wrappers. Programs may be written in Java, Perl or more specialized languages such as WebL [13]. Figure 3 shows a WebL program that looks up phones and addresses on the Yahoo People service.[2]

Writing these programs is a non-trivial task and requires programmer to closely inspect the HTML. If client-side scripts (*e.g.,* Javascript) are used, the programmer must also understand the logic behind the scripts. For example, if a site uses Javascript to process the values input in forms before they are submitted (*e.g.,* change all inputs to upper case), the wrapper must replicate this processing – otherwise, form submission may fail. In addition, details normally handled by browsers (*e.g.,* cookies) must be explicitly handled by these programs.

When complex (multi-step) navigation is required, problems are compounded, especially in the presence of dynamically generated pages. Because pages may change between the time a wrapper is created and its execution, wrappers must identify at each point during navigation, the *correct* action to execute. For example, the wrapper in Figure 3 returns a page with a list of links to people's names, their addresses and

---

[2]This example was modified from a sample from http://research.compaq.com/SRC/WebL.

4

```
<SMB>
<URL>http://people.yahoo.com/</URL>
<FORM>
  <action>http://phone.people.yahoo.com/
          py/psPhoneSearch.py</action>
  <ACTIONweight>25</ACTIONweight>
  <encoding>undefined</encoding>
  <ENCODINGweight>25</ENCODINGweight>
  <method>get</method>
  <METHODweight>25</METHODweight>
  <name></name>
  <NAMEweight>25</NAMEweight>
  <target></target>
  <TARGETweight>25</TARGETweight>
  <dom>opener.document.forms[0]</dom>
  <DOMweight>25</DOMweight>
  <property>use_stored_value</property>
  <ELEMENTS>
    <ELEMENT>
      <name>FirstName</name>
      <type>text</type>
      <property>use_stored_value</property>
      <required>true</required>
      <value>john </value>
    </ELEMENT>
    <ELEMENT>
      <name>LastName</name>
      <type>text</type>
      <property>use_stored_value</property>
      <required>true</required>
      <value>smith</value>
    </ELEMENT>
    <ELEMENT>
      <name>City</name>
      <type>text</type>
      <property>use_stored_value</property>
      <required>true</required>
      <value></value>
    </ELEMENT>
    <ELEMENT>
      <name>State</name>
      <type>text</type>
      <property>use_stored_value</property>
      <required>true</required>
      <value></value>
    </ELEMENT>
    <ELEMENT>
      <name></name>
      <type>reset</type>
      <property>use_stored_value</property>
      <required>true</required>
      <value>Reset</value>
    </ELEMENT>
  </ELEMENTS>
</FORM>
<LINK>
  <text>Adam John Smith</text>
  <TEXTweight>0</TEXTweight>
  <href>http://phone.people.yahoo.com/py/
        psPhoneEntry.py?firstname=Adam+John
        &lastname=Smith&street=12305+Lorien+Way
        &city=Oklahoma+City&state=OK
        &zip=73170-4724&phone=4056910981
  </href>
  <HREFweight>0</HREFweight>
  <target>null</target>
  <TARGETweight>0</TARGETweight>
  <dom>opener.document.links[17]</dom>
  <DOMweight>100</DOMweight>
</LINK>
</SMB>
```

Figure 4: SmartBookmark generated by the WebVCR to looks up e-mail addresses at Yahoo!

phone numbers. Suppose we want to extend this wrapper to follow the link to the first person in the list. Note that every time the script is executed for a different query, a different set of names will be returned. For example, for the "John Smith" query, the first link in the list is:

```
TEXT: Adam John Smith
URL: http://phone.people.yahoo.com/py/psPhoneEntry.py?firstname=Adam+John&lastname=Smith&
   street=262+School+St&city=Somerville&state=MA&zip=02145-2836&phone=6176232863
```

and for the "Mary Smith" query:

```
TEXT: A Marie Smith
URL: http://phone.people.yahoo.com/py/psPhoneEntry.py?firstname=A+Marie&lastname=Smith&
   street=&city=Grass+Valley&state=CA&zip=95945-0000&phone=5302734016
```

Since names and addresses will be different for the various executions, the script must specify the means by which the first link in the list will be identified. In this case, it could iterate through the list of links and return the first link that contains the substring psPhoneEntry.py?firstname.

Due to the complexity involved, writing programs to retrieve Web pages is out of reach for naive Web users who are not programmers. It is also not a feasible solution for applications which require the creation of a large number of wrappers. In order to address this problem, in [2] we proposed the WebVCR, a tool that automates to a large extent the creation of access wrappers. The WebVCR, transparently records a

user's actions as she browses through a series of Web pages. The sequence of actions, a SmartBookmark (SMB), can be saved and automatically replayed at a later time. Figure 4 shows an SMB to lookup people's addresses in People Yahoo. It consists of the following steps: go to the Yahoo People service; fill out "John Smith" in the `Telephone Search` form; and follow the first link ("Adam John Smith"). Note that an SMB stores information about each action performed by the user. For example, for links it stores not only the URL and text, but also the location and target. For forms, besides information about the actual form (*e.g.,* form name, action, etc.) it stores information about each of the form's elements. When a SMB bookmark is replayed, at each page, the WebVCR tries to identify the action that best matches the recorded action and executes the selected action – mimicking as closely as possible a user's actions (*e.g.,* if a Javascript handler is present in a submit button, it will be executed when the form is submitted).

In order to support different replay semantics, the WebVCR allows the user to control the matching of steps. One of the techniques used is *action weighting* – users can assign weights to the different attributes of links and forms. In the SMB of Figure 4, if we want to ensure that the first link in the list of people is retrieved, we can give 100% weight to the position and 0% to the link's text. This way, when the WebVCR is looking for a match, it will select the link in the retrieved page that has the same location and it will disregard the text. The reader is referred to [10] for more details about the matching algorithms used in the WebVCR.

Another important feature of the WebVCR is the ability to parameterize SMBs. Users may edit an SMB and change recorded input values. For example, for the SMB of Figure 4, we could directly edit the `FirstName` field and change its value, *e.g.,*

```
<name>FirstName</name>
<type>text</type>
<property>use_stored_value</property>
<required>true</required>
<value>mary </value>
```

During replay, the value **mary** would be used to fill out the `FirstName` field in the form. Parameterization is possible due to the robustness features of SMBs, and their ability to navigate through dynamic pages.

Two issues need to be resolved to support reliable parameterization of wrappers: internal attribute names that are undescriptive and invalid selections. Consider for example the *Book a flight* page at Travelocity, where users specify the itinerary details. The internal name for the departure month attribute is `dep_dt_mn_1`, which can be hard to identify. Also month values must have a specific format, *e.g.,* April is represented by "Apr", and if users input "April", the submission will fail. The WebVCR allows users to specify mappings from internal names to more descriptive tags of their choice. In addition, extra information is saved for elements (*e.g.,* all values in a selection list are saved) so that inputs can be checked for validity (as we discuss in Section 4, this additional information can be very useful for transcoding wrappers). Note that checking for validity of values is only possible for elements such as selection lists and radio buttons, where a domain is well-defined – it is not possible for text fields. Thus, even though we can reduce the likelihood of failures, they cannot be entirely avoided.

It is worth pointing out that parameterization only works reliably for deterministic sites – if there are different navigation paths for different values (or combination of values) of parameters, it will invariably fail when an alternate path is taken. For example, in DBLP[3] one can search for authors of papers in databases and logic programming by name. If the name is unambiguous, the result of the search will be a list of all that author's papers. But if multiple authors share that name, a list with these authors is displayed. To create a parameterized wrapper for DBLP, WebVCR would need the ability to express conditional statements.

---

[3]http://www.informatik.uni-trier.de/˜ley/db/indices/a-tree/index.html

2.2 Extraction Wrappers

Not only Web navigation but als individual Web pages are becoming increasingly complex. Consider the main Travelocity page. It has over 1300 lines of HTML, more than 100 links and almost 200 gif images. It is very hard to interact with such a page on a small device such as a Palm Pilot (one needs to scroll 16 times to view the full page), and for devices such as a Web-enabled mobile phone with a 3-line display, it is impossible.

Extraction wrappers can be used to extract individual elements in a page that are of interest to a user (or class of users), effectively filtering out irrelevant information. There are different kinds of extraction wrappers: some perform *coarse-grained* extraction, *e.g.,* extract an HTML table from a page; while others perform *finer-grain* and more structured extraction, *e.g.,* extract individual records from a table. Figure 5 illustrates the two modes.

Phone Search Results			Public Records Search Go
			People Locate Background Search

Showing 1 - 10 of 200
First | Previous | **Next** | **Last**   **Search Again**

Name (click for details)	Address	Phone (click to call)	
A Marie Smith	Grass Valley , CA	(530) 273-4016	Want more information? Get a "US Search"
A Mary Smith	344 Quivera St Dauphin Island , AL	(334) 861-6100	Want more information? Get a "US Search"
A Mary Smith	4 Dandiview Acres Seabrook , NH	(603) 394-8077	Want more information? Get a "US Search"
A Mary Smith	1550 S Roosevelt Rd 4 Portales , NM	(505) 276-8454	Want more information? Get a "US Search"
Aaron Mary Smith	5025 Independence St Maple Plain , MN	(763) 479-3812	Want more information? Get a "US Search"
Adam Marie Smith	22 Merlin Dr Fairfield , OH	(513) 858-3788	Want more information? Get a "US Search"
Al Maria Smith	311 Weldon Dr Watertown , NY	(315) 782-4551	Want more information? Get a "US Search"

(a) Coase-grain

```
<name_list>
<person>
  <name> A Marie Smith <name>
  <address> Grass Valley, CA </address>
  <phone> (530) 273-4016 </phone>
</person>
<person>
  <name> A Mary Smith <name>
  <address> 344 Quivera St, Dauphin Is-
land, AL </address>
  <phone> (334) 861-6100 </phone>
</person>
...
</name_list>
```

(b) Fine-grain

Figure 5: Different kinds of extraction

Fine-grained extraction outputs structured information which can be userful for applications that process the data (*e.g.,* a transcoder). However, this mode of extraction only works reliably for pages where the

7

information to be extracted has some structure. Many tools are available that perform fine-grained extraction [3, 5, 16, 13]. For example, W4F [16] defines a declarative language that allows programmers to specify extraction. LiXto provides a user interface where users can specify the desired extraction by pointing and clicking on page components – the system then automatically generates a set of rules (a script) to perform the extraction. For pages where there is no structure, or for which structure is not important, coarse-grained extraction can be applied. This is the approach we used in the WebViews system [12].

Important issues that must be handled for extraction wrappers include: how to identify elements to be extracted; how to deal with changes to Web pages. Different systems use different approaches. For example, LiXto wrappers use DOM addresses which are manipulated by the rules derived by the system. In WebViews, we use XPath [19] as the mechanism for specifying extraction expressions. XPath views an XML document as a tree and provides a flexible mechanism for addressing any node in this tree. One drawback of using XPath is its requirement that pages be well-formed. Since browsers are very forgiving in this respect, many Web sites generate pages that are ill-formed (*e.g.,* have overlapping tags, missing end tags, etc.). Consequently, the WebViews system must first *clean up* HTML pages (*e.g.,* using tools such as HTML Tidy [18]) before XPath can be applied. Although we considered using the XML DOM API for specifying extraction expressions, we selected XPath because it allows a more *flexible* and *easier* way to create robust clipping expressions that are immune to minor changes in page structure. DOM addresses (without storing extra information, or using other heuristics to compensate for page changes) can be very *brittle* even to minor layout changes.

Note that extraction wrappers can also be used to customize the presentation of the results to different devices. In the People Yahoo scenario, different wrappers could be created for different devices, *e.g.,* for a mobile phone, the wrapper could return the first address; for a Palm, the first three addresses could be returned; and for a laptop the complete table.

2.3 Correctness and Robustness

Usually, changes to Web pages do not pose problems to a user browsing the Web, but they do present a challenge to a system that performs automatic navigation. If a wrapper contains references to dynamic objects, *e.g.,* links that have embedded session ids, and forms which hidden elements that change from one interaction to the next, during navigation, the wrapper must locate the correct object (link, form or button) to be operated on, and this can be challenging in the presence of changes to Web pages (*e.g.,* addition/removal of banner ads). Moreover, any algorithm used to determine the new position of the object on the changed page must be reasonably fast, since it needs to be executed for every recorded user action. Hence, we cannot rely on algorithms that require expensive parsing or pattern matching (*e.g.,* [7]). As discussed in [2], during replay, if an exact match for a navigation action cannot be found in a page, heuristics (and optionally, users' hints) are an effective means to find the *closest match* for the action.

Extraction expressions also need to be made robust to changes to Web pages. Consider the following expressions that extract the first three itineraries at Travelocity:

```
(1) //html/body/center[2]/div/table[2]/tr/td/table[position()>=3 and position()<=8]

(2) //table/tr/td[(contains(string(),'Price:') or contains(string(),'Option')) and
       not(descendant::table)]/parent::tr/parent::table[position() >= 1 and  position() <= 6]
```

In the XPath expression (1) above, if the position of the `center` tag containing the desired tables changes (*e.g.,* a new preceding sibling `center` tag appears in the document), the expression will no longer retrieve the correct tables. Instead of absolute positions of nodes, the specification needs to include other information that helps the wrapper uniquely identify elements to be extracted, even if the node positions happen to

change. For instance, the XPath expression (2) specifies tables that contain the "Price" or "Option" string – this expression would still retrieve the correct itineraries even if new `center` tags are added.

Even though the heuristics and techniques can be developed to improve robustness [10], in some cases – if the page structure or navigation sequence changes radically – wrappers will break. Therefore, wrapper systems must be able to detect and report errors to the client, and ideally the wrappers should be easy to repair. In the WebVCR, if the system is not able to locate an object involved in a recorded action, it suspends the replay and notifies the user. The user may then re-record the SMB (to correct the problematic step) before the corresponding SMB is used again.

3   Delivering Web Clippings

As illustrated in Figure 2, in a thin-client environment, it is advantageous for wrappers to be executed in a Web server. Since processing (retrieval and extraction) is done in a powerful server with a fast connection to the Internet, communication between the client and the server is greatly reduced. This feature is especially useful in wireless environments where users must access the Web through high-latency, low-bandwidth connections, and through simple devices where some navigation steps (*e.g.,* those involving Javascript) are impossible to execute. Further, the task of transcoding the content becomes much easier – only a portion of the final page needs to be transcoded, and none of the intermediate pages.
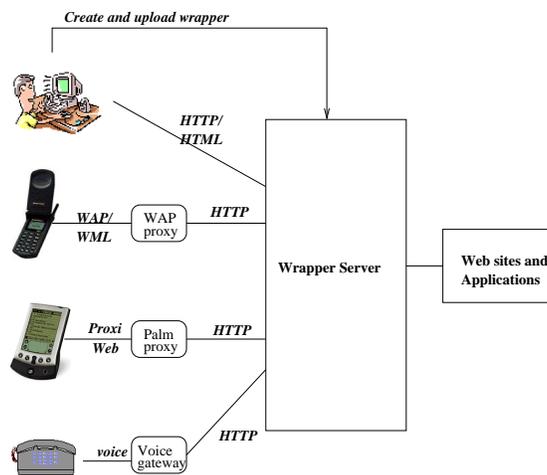

Figure 6: Wrapper-based content delivery architecture

By placing wrappers in a Web server, users may access them through virtually any browser. Figure 6 shows a general wrapper-based architecture for content delivery to diverse devices. Note that a wrapper server can be used in conjunction with gateways that perform protocol conversion to and from HTTP, as well as the necessary transcoding of content retrieved by the wrapper, *e.g.,* a WAP proxy to allow access to WAP-enabled devices, a Voice gateway to enable voice access to Web content, and even specialized gateways such as a Palm proxy.

Some useful features of a wrapper API include:

- The ability to *specify values* for parameterized wrappers, *e.g.,* the password to access a bank account.

- Specify the *mode of delivery*, pull or push.

9

- Specify whether the results of a wrapper should be *cached* and how often it should be *refreshed*.

Given the Web's unpredictable behavior (network delays, unreachable sites, etc.), caching plays an important role in that it gives users instant access to information. Users should be able specify for each wrapper, if and how often it should be executed and cached (*e.g.,* execute the Travelocity SMB every 24 hours, and cache the itineraries). In addition, users may also specify how they want the wrapper to be delivered. In *pull mode*, the URL invokes a CGI script at the server, which in turn executes the wrapped specification and immediately returns the clipped content to the requesting client. In *push mode*, the execution and delivery of the wrapper are asynchronous, *i.e.,* the wrapper can be returned to the client later, possibly through protocols other than HTTP (*e.g.,* clippings could be emailed). Push mode is preferable when back-end Web sites are slow or temporarily unreachable, or when the end user cannot or does not want to keep a session open for too long. (Some wireless data services, such as Sprint PCS, charge for usage time.)

4 Transcoding

By simplifying the content retrieval process, and filtering out uninteresting components of Web pages, a wrapper-based architecture for content delivery can greatly simplify transcoding and render the desired information in far more user-friendly manner than general transcoders can. Whereas a loosely-coupled approach is possible where generic transcoders process the content output by a wrapper, the transcoding functionality can also be incorporated into the wrapper server, with the added advantage that the user can now *annotate* a wrapper and supply extra information that can be used in the transcoding process to produce better quality content.

Consider for example transcoding information in an HTML table into voice. A table may be organized row-wise, column-wise, or neither (*e.g.,* being used simply for layout) – and each requires substantially different transcoding. For instance, if the table of interest in Figure 5(a) is treated as being there simply for layout, the transcoded voice would be partially incomprehensible—something like: "Showing 1 -10 of 200 First Previous Next Last...". The knowledge that Name, Address and Phone are columns to transcode and that the table is laid out row-wise allows the transcoder to pair headers and values together and to eliminate uninteresting data.

Also, information may be available in the wrapper specification that can aid in transcoding. For example, SMBs store the list of acceptable choices about entities like radio buttons and pull-down lists. Not only does this information allow the system to prevent bad user-input, it also makes it possible to transcode the parameterized data in a more convenient form. If such a form needs to be read out to a user, rather than reading out each possible value with an associated number and having the user enter that number by voice or touchtone, the system can generate a grammar that accepts the legitimate choices. The lists are generally small enough that voice recognition will work reasonably well. For example, when giving a user the choice to enter the name of a state, allowing the user to say "WV" immediately is far more convenient than asking her to wait to hear and respond to "47 WV".

5 Discussion

We have given an overview of Web wrappers and discussed how they can be used to provide ubiquitous access to Web content. Wrapper-based solutions to content delivery can be used in many different scenarios:

- *ISP/WSP:* ISPs and WSPs may create wrappers that give their users personalized access to content published by a third-party content provider;
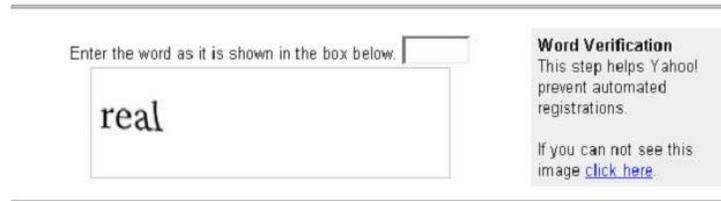
Figure 7: Signing up for Yahoo! mail

- *End-user:* an end-user may create her personal wrappers;

- *Content provider:* content providers such as CNN and Yahoo! may provide wrappers that give customized access over their content;

- *Enterprise:* an enterprise may create wrappers that export device-specific views for some intranet services.

These different application scenarios have different requirements and directly impact the choice of techniques for creating access and extraction wrappers. For example, since in the ISP/WSP scenario wrappers are created over third-party content that change constantly, wrapper robustness is critical. In contrast, robustness is less important in an Enterprise scenario, where wrappers are built over proprietary and often more stable (and static) content.

Wrappers, however, are not a panacea. Although they offer a good alternative to export some Web content to diverse devices, they are not the final answer to the ubiquitous Web problem. Since, in general, wrappers are not guaranteed to work, using wrappers for important transactions (*e.g.,* for writing checks or performing bank transfers) is not a good idea. In addition, even though solutions exist to wrap complex Web sites, sites – sometimes intentionally – make it virtually impossible to automate access to their content. A good example is Yahoo Mail. In order to sign up for a new Yahoo mail account, the user is forced to read a distorted string (see Figure 7) and type the string into a form field – a true wrapper stopper!

References

[1] http://www.amazon.com/anywhere.

[2] V. Anupam, J. Freire, B. Kumar, and D. Lieuwen. Automating Web navigation with the WebVCR. In *Proc. of WWW*, pages 503–517, 2000.

[3] N. Ashish and C.A. Knoblock. Wrapper generation for semi-structured internet sources. *SIGMOD Record*, 26(4):8–15, 1997.

[4] T. Ball and et al. Sisl: Several interfaces, single logic. *International Journal of Speech Technology*, 2002. To appear.

[5] R. Baumgartner, S. Flesca, and G. Gottlob. Visual web information extraction with lixto. In *Proc. of VLDB*, pages 119–128, 2001.

[6] CDPD. http://www.wirelessdata.org/develop/cdpdspec.

[7] H. Davulcu, G. Yang, M. Kifer, and I.V. Ramakrishnan. Computation aspect of resilient data extraction from semistructed sources. In *Proc. of PODS*, 2000.

[8] N. Dunn and C. Rumble. http://www-4.ibm.com/software/developer/features/feat-transcoding.html, 1999.

[9] A. Fox and E. Brewer. Reducing www latency and bandwidth requirements by real-time distillation. *WWW5/Computer Networks*, 28(7-11):1445–1456, 1996.

[10] J. Freire. Creating robust access wrappers: Techniques and experiences. Technical report, Bell Labs, 2002.

[11] J. Freire and B. Kumar. Web services and information delivery for diverse environments. In *Proc. of VLDB Workshop on Technologies for E-Services*, 2000.

[12] J. Freire, B. Kumar, and D. Lieuwen. Webviews: accessing personalized web content and services. In *Proc. of WWW*, pages 576–586, 2001.

[13] T. Kistlera and H. Marais. WebL: A programming language for the Web. *WWW7/Computer Networks*, 30(1-7):259–270, 1998.

[14] http://channel.nytimes.com/partners/palm-pilot.

[15] ProxiWeb. http://www.proxinet.com.

[16] A. Sahuguet and F. Azavant. Building light-weight wrappers for legacy web data-sources using W4F. In *Proc. of VLDB*, pages 738–741, 1999.

[17] http://www.sprintpcs.com/wireless/ wwbrowsing_providers.html.

[18] http://www.w3.org/People/Raggett/tidy.

[19] http://www.w3.org/TR/xpath.