WebContext: Remote Access to Shared Context

Robert Capra, Manuel A. Pérez-Quiñones, Naren Ramakrishnan
Department of Computer Science
Virginia Tech
660 McBryde Hall (0106)
Blacksburg, VA 24061
rcapra@vt.edu, {perez, naren}@cs.vt.edu

ABSTRACT

In this paper, we describe a system and architecture for building and remotely accessing shared context between a user and a computer. The system is designed to allow a user to browse web pages on a personal computer and then remotely make queries about information seen on the web pages using a telephone-based voice user interface.

Keywords

Shared context, voice user interfaces, information access, telephone-based user interfaces, software architecture, VoiceXML.

1. INTRODUCTION

Common ground refers to experiences and knowledge shared among participants in an activity [19]. In his book, *Arenas of Language Use*, Herbert Clark describes common ground in terms of shared information:

> Common ground is a type of shared information. The common ground between Ann and Bob, for example, is the sum of their mutual knowledge, mutual beliefs, and mutual suppositions... [7]

In the book, Clark makes the argument that conversation between two participants cannot occur without the accumulation and use of common ground. In our work, *shared context* refers to the accumulated common ground between the computer and the user as a result of human-computer interaction.

In today's world, people commonly spend several hours a day interacting with personal computers (PCs), personal digital assistants (PDAs), and cell phones. Despite hours of interaction, current software makes use of very little context from interacting with users. Additionally, context established on one computing device is often difficult to make use of on a different device. Remote access to context is also difficult. This increasing need for portable, cross-platform, remotely accessible shared context between users and computers is a central topic of our research.

We are especially interested in the portability and accessibility of shared context. Shared context needs to be accessible from different environments that have different input/output modalities: personal computers with a keyboard, mouse and large screen; PDAs with small screens and an input stylus; cellular telephones with a small screen and buttons; and voice interfaces for access from any telephone.

We have developed an architecture for capturing, storing, accessing and using shared context across different computing environments. This architecture allows context to be captured in one environment (a personal computer) and then remotely accessed from a different environment (a telephone-based voice user interface).

Using this architecture, we have implemented a system called WebContext. WebContext allows a user to browse web pages on their personal computer and then make queries about information viewed on those web pages using a voice user interface.

One of the interesting problems in creating such a system is that the speech recognition component of the system must be configured to understand spoken references to a dynamic set of information. We will describe this problem in more detail in section 5.2.3.

2. Existing Uses of Context

The primary two ways that current personal computer applications and operating systems store context is by saving documents and application preference settings. Recently, personal computer operating systems have begun capturing and using additional parts of a user's interaction with the system. For example, Microsoft Windows and the Apple Macintosh OS both have introduced a feature that stores "recently used documents" in a special location for easy access by users. For many years Unix operating system command shells have had an extensive interaction history tool (the "history" command) that allows users to review and re-issue commands.

Most PDAs provide mechanisms for synchronizing data between the PDA and other devices, but they are sometimes limited in the types and amount of data they store and exchange. For example, many PC applications must have special synchronization software installed and configured to allow them to share data with PDAs. For some applications, no such capability exists.

PUI 2001, Orlando FL, USA.

Many cell phones retain information about interaction histories by saving phone numbers. They store the phone numbers of people who you recently called, of people who recently called you, of calls you missed, etc. However, cell phones too are limited in the types and amount of data they utilize and exchange with other devices.

Much of the interaction history and context that is captured on these devices (PCs, PDAs, and cell phones) is used only at a surface level. Ordered lists of recent events such as phone calls, documents opened, and preference settings can be useful and do not require much analysis to construct. However, we believe that more robust approaches may lead to more streamlined user interactions and increased usability.

To create more robust user interfaces that make use of a high degree of shared context with users, interfaces need to capture contextual information, integrate multiple sources of information, and perform analysis on available information. Rich models of context and content are required. For example, instead of just placing recently dialed phone numbers on a "recently called" list, a cell phone interface could consider additional factors such as: Was the call completed? How long did the call last? What time of day was this call placed? Is this a phone number that is called frequently? Is this phone number in the user's dialing directory?

The work we present here on the WebContext system is part of our initial efforts to develop systems that make robust and integrated use of shared context to improve user interfaces. In this work, we have focused on using the content of web pages browsed by the user to build a model of shared context. The idea here is that the computer "observes" everything that the user sees in the browser window. This collaborative experience between the user and computer builds shared context. In future work, we plan to extend this approach to integrate contextual information from multiple applications.

3. Usage Scenario

To illustrate the architecture and use of WebContext we will use the following scenario throughout the paper. Although this scenario contains fictitious places and names, the current version of the WebContext system performs similarly on actual web pages.

Anytown Hotel Home Page

1234 Main Street
Anytown, USA 12345
Phone: (800) 555-1234
Fax: (800) 555-5678

The Anytown Hotel is a charming 50 room hotel located in the heart of downtown Anytown, USA. The hotel is known for the seafood restaurant on the lower level.

Home | Reservations | Rooms | Directions

Figure 1. Anytown Hotel Web Page.

Anytown Hotel Scenario

Mary decides to plan a short vacation for the upcoming weekend. Using her home computer and web browser, she decides on a destination for her trip and makes a hotel reservation on-line. (The home page for the hotel she selected is shown in Figure 1.)

Later, Mary decides to call the hotel to check the reservation. Instead of logging back onto the Internet to find the phone number of the hotel, Mary calls into the WebContext system using her cell phone:

[1] Mary:	<calls into the WebContext system>
[2] *System*:	*Welcome to the WebContext system. Please say some words to help identify the pages to search.*
[3] Mary:	<remembering the name of the hotel and that it has a seafood restaurant> "Anytown hotel and seafood"
[4] *System*:	*What piece of information are you looking for?*
[5] Mary:	"the phone number"
[6] *System*:	*Looking for phone numbers on web pages with Anytown Hotel and seafood. Is this correct?*
[7] Mary:	"Yes"
[8] System:	*Now looking for matches. <pause>* I found a total of two phone numbers on one page. On the page titled "Anytown Hotel Home Page," the first result is "phone, eight hundred, five five five, one two three four." The next result on this page is "fax, eight hundred, five five five, five six seven eight". That is the end of the results.

Figure 2. WebContext Sample Dialog.

The current implementation of the WebContext system supports dialogues such as the one shown in Figure 2. It allows users to make queries – using a voice user interface – about pieces of information such as phone numbers and addresses that they have seen on previously browsed web pages. In Section 6, we present a step-by-step description of how the WebContext system processes the Anytown Hotel Scenario sample dialog.

4. Related Work

In this section, we discuss several areas of related work: web companions, voice interfaces to the web, and computer access by phone.

4.1 Web Companions

Web companions are programs that monitor a user's or group of users' browsing habits and try to use this information to make recommendations of other web sites of interest to the user. Lawrence [14] gives descriptions of many projects that have attempted to incorporate contextual information into web searching.

Many of these systems observe users' behaviors and/or environment and use this information to help recommend or predict web pages of interest (WebWatcher [12], Personal WebWatcher [20], Letiza [15], "Let's Browse" [16], WebMate [5]). Some systems use contextual information to automatically fetch (Watson [4]) or index (Vistabar [17]) web pages for users.

The *Haystack* project at MIT has focused on creating personalized information repositories for individuals. Haystack automatically gathers information, adapts to its user, and tries to use information gathered to help organize and find data [1].

Our work shares concepts of these projects in observing and trying to make use of a user's web browsing context. In the current implementation of WebContext, we have focused on exploring the use of shared context in a different environment (remote access over a telephone) and the use of a different interface modality for access (a voice user interface).

Browser supported bookmarks (favorites) and on-line bookmark managers such as Backflip[1], Blink[2], Yahoo! Bookmarks[3], (as well as other bookmark services) attempt to help users create indexes of web pages they wish to "remember" and possibly share with other people or access from other computers. However, these tools often require users to perform explicit actions to add a site to their bookmark list and may require additional effort to classify and maintain the list of bookmarks.

4.2 Voice Interfaces to the Internet

A number of voice interfaces to Internet content and web browsing have been developed in recent years. Current voice interfaces to the web fall into two main categories: 1) voice interfaces to screen displays and 2) voice-only interfaces (typically using a telephone as an input and output device). Christian, Kules, Shneiderman, and Youssef provide a good summary of a number of these systems in their comparison of using voice versus mouse to control web browsing [6]. Another source of information about voice web browsers is the web site of the 1998 W3C Workshop on Voice Browsers [24].

PhoneBrowser is a system developed at Lucent Technologies for browsing unmodified HTML web pages using only a telephone. It uses text-to-speech to read page content to users and uses speech recognition to allow users to control the browser [3].

Conversa Web is a voice-controlled web browser that was created through extensions to Internet Explorer. It allows users to speak the text of links and use navigation commands (such as "go back") to browse web pages [21]. Charles Hemphill conducted earlier work on a voice-controlled web browser at Texas Insturments [10].

Our WebContext system provides access to information extracted from web pages. However, it does not attempt to support full web browsing by voice. Instead, it focuses on helping users quickly retrieve pieces of information they have previously viewed on web pages.

VoiceXML [23] is an XML-based language that was developed by the VoiceXML Forum and submitted to the World Wide Web Consortium (W3C). VoiceXML provides a standard language for developing voice-enabled Internet content and applications. It is a language similar to HTML that can be browsed using VoiceXML voice browsers. The WebContext system currently uses VoiceXML to implement the part of the interface that allows users to make remote queries by voice.

4.3 Computer Access by Phone

Several projects have focused on creating telephone-based interfaces to computer resources. *Phoneshell* is a system developed at MIT that supports "remote voice access to personal desktop databases such as voice mail, email, calendar, and rolodex" [22] using a touch-tone telephone interface. *Chatter* (described with Phoneshell in [18]) provides functionality similar to Phoneshell and uses speech recognition to support a voice user interface.

SpeechActs is a system developed by Sun Microsystems' Speech Applications Group that allows users to use a telephone-based voice user interface to interact with desktop applications such as Sun's Mail Tool and Sun's Calendar Manager. The SpeechActs system was designed with shared context and conversational principals in mind [25].

Personal Assistant services such as *Wildfire*[4] and General Magic's *Portico*[5] allow subscribers to remotely access voice mail, email, and schedule information using a telephone-based voice user interface.

Our work on WebContext is related to this previous work on remote access to computer resources. We have focused on developing interfaces to access shared context built from previous dialogs. Initially, we are exploring telephone-based access to context built from web browsing sessions on a PC. In this respect, our voice interface is a complement to the shared experience of the user and the web browser.

5. WebContext Implementation

In this section we will describe the query model and details of how the WebContext system was implemented (including the architecture we developed for making shared context remotely accessible). We also describe the query application component that can be used to allow users to access their shared context from any telephone using a voice user interface.

5.1 Query Model

WebContext uses a simple model for building context and for its query capabilities. In our current version, shared context is built exclusively from the content of web pages viewed by users. A web page, *P*, is viewed as consisting of two parts: 1) a set *I* of *extractable information pieces* and 2) a set *C* of *context indicators*.

Extractable information pieces are logical groups of information

[1] http://www.backflip.com

[2] http://www.blink.com

[3] http://bookmarks.yahoo.com/

[4] http://www.wildfire.com

[5] http://www.genmagic.com/products/portico.shtml

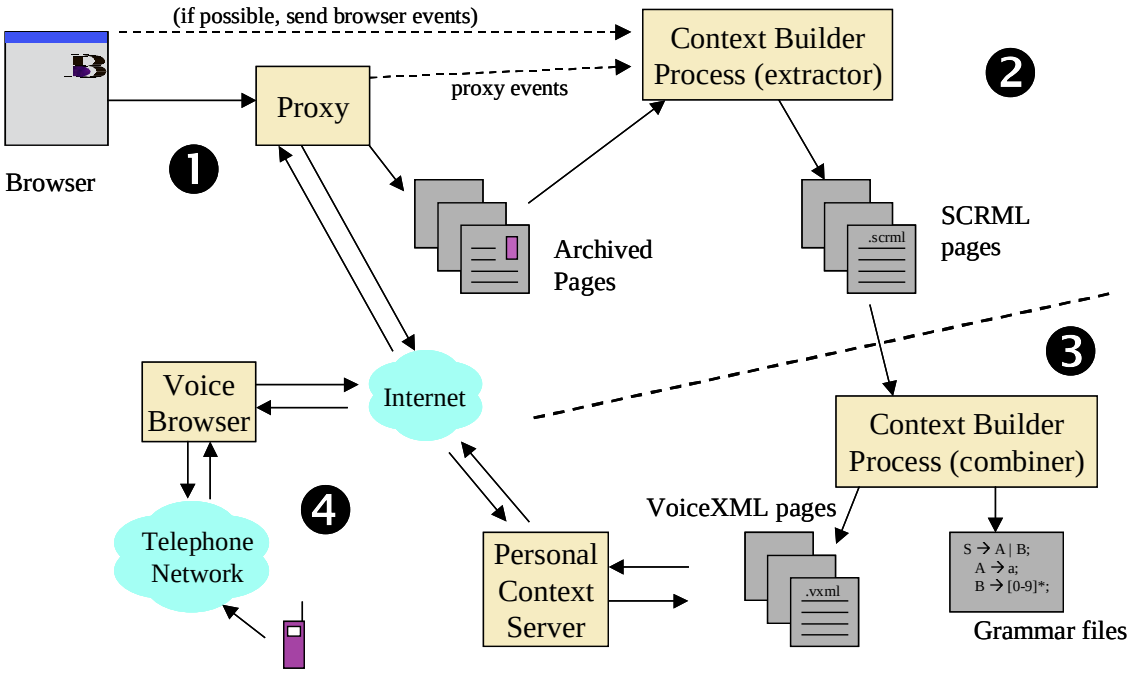

Figure 3. WebContext Architecture for Building and Accessing Shared Context

or HTML text that can be extracted from the page. These are items that a user might wish to refer back to at a future time. Initially, we have implemented simple extraction modules for three types of information: phone numbers, addresses, and dates. We intentionally selected these information types because it is possible to construct simple, yet usable, extraction modules for them. In our initial work, we wanted to focus on developing an architecture for capturing and using shared context. The next step is to investigate particular algorithms, tailored to specific types of information resources.

Context indicators help identify a particular context. Currently, context is being modeled by the textual content of the web pages, so the context indicators are taken from the HTML of the web page. In other words, we are using some of the text on a web page as an indicator of context. The details of this process are described in section 5.2. In the future, we plan to investigate including additional indicators such as how long and how many times the user viewed this page, what link brought the user to this page, and what other pages were also viewed around the same time as this page. It is interesting to note that $I \cap C$ is not necessarily null, meaning that information pieces can be context indicators and vice-versa.

A well-formed query consists of at least one context indicator and one extractable information piece. In the Anytown Hotel scenario sample dialog (see Figure 2), Mary provides two context indicators, "Anytown Hotel" and "seafood" (line 3). The system explicitly asks for what information piece should be retrieved within the stated context (line 4). Mary requests one extractable information piece, "the phone number" (line 5). The context indicators help establish common ground for the query between Mary and the system. They indicate to the system that the she is interested in talking about web pages that meet the criteria specified by the context indicators (in this case, pages that contain the phrases "Anytown Hotel" and "seafood"). By asking for the information piece "the phone number," Mary is requesting that the system return any information about phone numbers on web pages that meet the criteria of the context indicators.

In many ways, this is similar to what a person might do in asking another person for a piece of information. Consider the following fictitious dialogue between two people:

[1] R: Do you remember the poster we saw about the jazz concert?
[2] P: Yes.
[3] R: What date was it?

In order for a successful collaboration to take place, the requestor (R) tries to establish common ground with the provider (P) by saying the phrases "poster" and "jazz concert" to help the provider recall the correct context for the query (line 1). Then, after establishing common ground in line 2, the requestor can make information requests within that context in line 3.

Although this query model is fairly simple, it provides a good starting point for investigating user interfaces that incorporate shared context built in different environments. Expanding the query model is one of our goals for future work.

To help evaluate the user interface to WebContext, we conducted some informal usability testing with members of our research group. Based on this testing, we have observed that users may not make a distinction between context indicators and information types. Because of this, we are re-designing the user interface of the query application to remove this distinction from

```
01  <?xml version="1.0"?>
02  <scrml version="0.1">
03
04  <contextobject type="html"
05     name="anytown.html"/>
06
07  <referingobject name="unknown"/>
08
09  <group name="H1">
10     <H1> Anytown Hotel Home Page  </H1>
11  </group>
12
13  <group name="title">
14     <title> Anytown Hotel Home Page</title>
15  </group>
16
17  <group name="bold">
18     <bold> Phone:  (800) 555-1234  </bold>
19     <bold> Fax:  (800) 555-5678  </bold>
20     <bold> heart of downtown </bold>
21  </group>
22
23  <infogroup name="phone">
24     <phone> Phone:  (800) 555-1234 </phone>
25     <phone> Fax:  (800) 555-5678 </phone>
26  </infogroup>
27
28  <infogroup name="address">
29     <address> 1234 Main St </address>
30  </infogroup>
31
32  <group name="bodytext">
33     <bodytext>
34  Anytown Hotel Home Page      Anytown Hotel
35  Home Page   1234 Main Street Anytown, USA
36  12345   Phone:  (800) 555-1234     Fax:
37  (800) 555-5678     The Anytown Hotel is a
38  charming 50-room hotel located in the
39  heart of downtown  Anytown, USA. The hotel
40  is known for the seafood restaurant on the
41  lower level.  Home | Reservations | Rooms
42  | Directions
43     </bodytext>
44  </group>
45  </scrml>
```

Figure 4 – Sample SCRML file

the user's perspective (the system still uses this distinction). Additionally, we are trying to make the interface more of a collaborator in the process by allowing users to say as much or as little input as they wish at the initial prompt. If the system needs additional information to fulfill a request or refine a search, the interface will then take a more active part in moving toward a shared context with the user.

5.2 System Architecture

Figure 3 gives an overview of the system architecture for the WebContext system with four major parts identified: 1) capturing context, 2) building context models through information extraction, 3) combining context from multiple sources, and 4) using context in a different environment.

Currently, we have accomplished Part 1 by archiving web pages by hand. Parts 2 and 3 are central parts of the WebContext system, can be run on a user's computer, and are what make the shared context available to other applications. Part 4 is the query application that provides a voice user interface to make remote queries about information in the shared context. In the following sections, we will describe each of these components.

5.2.1 Capturing Context (1)

The first part of the architecture deals with capturing and archiving events as the user interacts with the web browser. As the user browses web pages, a proxy server can be used to archive all the pages viewed and requested by the user. This proxy server could be local to the user's machine (in order to build and store the context locally), or could be located elsewhere.

In our current view, context would be stored locally on the user's machine in order to give the user greater control over the context. This requires that the user's machine be connected to the Internet and have sufficient disk space to store the shared context. Recent estimates of the average size of web pages range from about 10 to 60 kilobytes [9],[2]. Based on these estimates, it is not unreasonable to imagine current personal computers archiving all the web pages a person views. If we assume a 100k average web page size (larger than these estimates) and estimate 40 page views per day, it would take about 13.7 years to fill a 20Gb hard disk. That's a lot of context!

For the shared context models that we build in the current WebContext system, only textual information from the web pages is stored. Disk space is not required to store pictures and graphics on the pages. Archived pages could be stored in a database and indexed at periodic intervals, such as overnight. In the initial version of WebContext, we simulated this part of the system by saving web pages by hand into a common directory and then running the context building routines on all the HTML files saved.

There are many valuable pieces of information about the context in which a user is browsing that cannot be determined from looking only at the web page requests as recorded by a proxy server. To gain additional context information, a web browser could be configured to send browser events to the context building components of the system through the use of browser plug-ins or extensions. Several recent projects have used or proposed using measures such as length of time to view a web page [17] and estimates of mouse movements [11] to help give feedback to a system that is trying to learn user browsing preferences. Our initial version of the WebContext system does not make use of any "browser events", but we plan to explore including them in future work.

5.2.2 Building Context Models (2)

The second part of the WebContext architecture is focused on building models of context based on the archived web pages that the user browsed (i.e. the pages collected by the proxy server).

Currently, we model context by extracting information out of the archived HTML pages. An extractor program (written in Perl) with modules for extracting various information pieces is run on each HTML page in the set of archived pages. For each page, the extractor produces a counterpart XML document that represents context indicators and information pieces found on that page. The XML document is stored in a simple XML-based

specification language we have been developing to help represent context. For convenience in this paper, we will refer to this as the Shared Context Representation Markup Language (SCRML). This representation is in the early development stages and is still evolving.

The extractor program looks for two major types of data in the HTML pages: extractable information pieces and context indicators. Information pieces are things like phone numbers, addresses, and dates that the extractor has a module to identify and extract. Context indicators are items on the page that help identify it and related pages. The title of the page, words that appear in links or in bold type, and headings can all be used as context indicators. The body text of the page is also treated as a context indicator and is used in later processing stages to help build a language model to allow the user to speak about words and phrases they saw on the page.

The SCRML page contains information pieces found by the extractor modules, context indicators found by the extractor main program, and additional information that can be determined about the page or that has been provided by the proxy server or browser. Figure 4 shows an example of the SCRML generated from the Anytown Hotel web page example (Figure 1).

5.2.3 Combining Context from Multiple Sources (3)

The Combiner module is at the heart of the third part of the WebContext architecture. The role of this module is to combine information contained in a set of SCRML pages into a grammar that can be used by another application (in our case the VoiceXML query interface). This grammar helps provide access to the shared context for other applications. In future work, the combiner may provide additional "views" to access the shared context. For example, the combiner might use the <referringobject> information in the SCRML pages to create a graph of how pages were browsed or to group related web pages together.

To make use of the shared context in the voice interface, users need to be able to talk about items they saw on the web pages. To accomplish this, the speech recognition component of the system must be configured to be able to recognize spoken references to items that were on the web pages. The initial version of WebContext was implemented using a speech recognizer that is configured using a context-free grammar (CFG) as the language model. The use of a CFG-configured speech recognizer presents some limitations about the how the language model can be constructed and presented to the recognizer. We are aware of limitations in the use of CFGs for our application. Nonetheless, in our initial implementation, we chose to use a speech recognizer configured by CFGs because they are easily accessible, are supported by current VoiceXML interpreters, and we had existing expertise in using these types of recognizers.

Statistical language models (such as N-grams) can provide a great deal more flexibility and power for recognition tasks such as the one that we are attempting to address in the WebContext system. We are investigating several options for incorporating a speech recognition system that will meet our telephony needs as well as provide an ability to be configured using statistical language models. This may require moving our interface implementation away from VoiceXML (to the best of our knowledge, current VoiceXML systems all use recognizers configured by CFGs).

There are two parts to the CFG grammar we are currently using: a fixed part that contains elements that do not change, and a dynamic part that is created by the Combiner module based on the contents of the SCRML files. The format used for the grammar is the Java Speech Grammar Format version 1.0 [13].

The Grammar Inclusion Problem

Determining what terms and combinations of terms from the web pages to include in the dynamic part of the grammar is a tricky problem. We call this the "Grammar Inclusion Problem." It is important to have good coverage of the terms that users will expect to be able to say, or else the system will not be as useful (or as usable). However, if all the terms and combinations are included, the grammars may become prohibitively large. Statistically based language models should help overcome aspects of this problem, but we are also exploring techniques for determining what items are important to include in the grammar/language model and what items are not as important to include.

In our current prototype, we use a fairly simple approach – we include in the grammar all unigrams, bigrams, and trigrams of words from the titles, headings, and bold words on the pages. Unigrams are all the single words, bigrams are all the two-word sequences, and trigrams are all the three word sequences. We also include all unigrams from the text of all the pages (after filtering out HTML tags and stop words).

This simple approach can require tradeoffs when trying to scale up to larger numbers of pages. In one sample of 15 web pages, when we included the unigrams, bigrams, and trigrams from all the non-HTML text on the page, the resulting grammar contained approximately 11,500 phrases. For the same set of pages, when only the unigrams, bigrams, and trigrams from the text in the titles, headings, and bold words were used, the resulting grammar had approximately 250 phrases. However, this reduction in the grammar size also greatly reduces the percentage of items on each page to which the user can make references.

Modern speech recognition systems can handle grammars with thousands of phrases and retain good accuracy (less than 10% error rates) [8]. However, more robust techniques are needed to allow the system to construct grammars that can handle hundreds of web pages. Again, this is where statistical language modeling techniques should help.

We have considered several techniques to determine what items to include in a CFG. We believe these techniques can also be used to help influence a statistical language model. One technique is to use information about how recently a page has been accessed. For example, a user might preface a query with a time frame such as, "web pages seen in the past week." The system could then dynamically adjust the grammar to include only the terms relevant to that time frame. Alternatively, the system could automatically adjust the grammar so that more recent pages are always in the grammar, but for older pages, the user must specify a time frame.

Another approach is to use techniques such as term-weighting and frequency counts to help decide what terms are the most relevant to include in the grammar. These techniques have been used in web page classification and information retrieval systems (see [16]).

An important aspect of evaluating the effectiveness of any of these approaches will be to compare the predicted terms against what users actually remember and say from the web pages when they make verbal references. We anticipate that users may not remember the exact terms on the pages, so techniques to match related terms may be needed to improve system performance. It is also possible that users may wish to get access to certain information that they do not remember enough context about to form an initial query. In these cases, the user interface will need to work in a collaborative fashion with the user to help establish the shared context if possible.

5.2.4 Using Context in Different Environments (4)

The fourth part of the WebContext architecture is the access mechanism for making the shared context available in other applications. In this project, we focused on accessing shared context through a voice user interface over a telephone.

VoiceXML is an emerging language for creating voice user interfaces that can be accessed over the Internet [23]. It is based on a client-server model similar to current web applications. Like HTML pages, VoiceXML pages are made available on standard HTTP web servers. Instead of web browsers on the client side, VoiceXML pages are browsed using voice browsers. There are voice browsers available that allow users to interact with VoiceXML content in several different ways: 1) by calling in on a telephone, 2) using a computer equipped with a sound card, speakers and microphone, and 3) using a text interface where the user types in their input on a keyboard and the browser renders the VoiceXML output prompts in a textual format.

To provide users mobile access to shared context, we envision users connecting to a "personal context server" running on a computer that is always connected to the Internet. This computer could be a computer at home or at work, or could be a shared system that serves contexts for many users. In our current WebContext system, this server requires access to four sets of files: 1) VoiceXML pages that implement the user interface, 2) CGI scripts that process the forms and implement the application logic, 3) grammar files generated by the Combiner, and 4) SCRML files generated by the Extractor. The VoiceXML files and CGI scripts are part of the voice query application, while the Grammar files and SCRML files are dynamically generated based on the web pages the user has been browsing. The SCRML files are used by the CGI scripts as a database of information to be searched.

6. Step-by-Step System Interaction

Below is an explanation of the sequence of steps the system goes through for the Anytown hotel scenario described in Section 3:

Step 1: *Mary calls into the WebContext system*
At this step Mary calls into a voice browser using a telephone. This voice browser could be running on one of her computers if she has the necessary telephone equipment installed on her computer, but more likely it is a voice browser provided to many users by some other entity. For example, businesses could provide access through a toll-free number to a voice browser for employees to access their information.

Access could also be achieved through a voice portal. Voice portals are similar to web portal sites such as Yahoo! or Excite except that instead of helping users find web content, they help users access VoiceXML content. Several companies such as TellMe (www.tellme.com) and BeVocal (www.bevocal.com) are offering voice browsing services with toll-free access numbers.

We have obtained specialized telephony hardware and speech recognition software for our lab that allows us to build voice user interfaces that can be accessed through a telephone. However, we do not yet have our telephony hardware and speech recognition system working with a VoiceXML browser. Thus, in the development of WebContext, we have simulated calling in on a telephone by using a VoiceXML development kit that supports voice browsing using a microphone and speakers attached to a personal computer. This development kit allows us to write applications in the same VoiceXML code that would be used to run the application over the phone, but we can test them using the built-in audio of a PC workstation.

To simulate the action of Mary accessing her personal context server, we start the voice browser with the VoiceXML page of the query application. This page is located on a server with access to the shared context.

Step 2: *The system plays a greeting and prompts Mary for some words to help identify the pages to search.*
The greeting and prompt are part of the instructions contained on the initial VoiceXML page for the query application.

Step 3: *Mary says "Anytown hotel and seafood"*
The voice browser acts on the code of the VoiceXML page and attempts to recognize what Mary said based on the grammar that was built by the Combiner module. Both these phrases are part of the grammar since Mary visited a web site with these terms. Our initial implementation does not support very robust error handling, so if the utterance is not in the grammar, it will simply re-prompt the user for the information.

Step 4: *The system asks what type of information to return*
This sequence of questions is part of the VoiceXML code and is part of a VoiceXML form that is being filled.

Step 5: *Mary responds, "the phone number"*
There is another grammar that is associated with the information types that can be requested. This grammar includes phrases for the information types currently processed by the system: phone numbers, addresses, and dates.

Step 6: *The system plays a prompt to verify the information it has gathered so far.*
This is a common practice at the end of requesting a series of information from the user in a voice interface. The system repeats information that it heard and asks the user if the information is correct. If the user responds affirmatively, the system proceeds to process the request. If the user gives a negative response, the system will back up and re-prompt the user for the information.

Step 7: *Mary responds affirmatively ("yes")*
This response causes the VoiceXML code to branch to a section of code that will play a prompt letting the user know that the system will start the search.

Step 8: *The system plays a prompt, "Now searching"*
After this prompt is played, the VoiceXML code submits the responses it has gathered from the form fields (the information type and the words to identify the pages to search) to a CGI script on the server for processing. After submitting the form, the voice browser waits for a response from the server.

Step 9: *The system returns and plays the results*
The CGI script on the server uses the information submitted by the initial VoiceXML page to create and execute a query. In the Anytown Hotel scenario, the terms "Anytown hotel" and "seafood" are used to narrow the list of web pages to search.

Next, the CGI script looks for the requested pieces of information (in this case, phone numbers) on these pages. The information has already been extracted and stored in an organized form in the SCRML pages, so instead of looking for the information on the original HTML pages, it searches the counterpart SCRML pages. This is one of the purposes of the SCRML pages – to make it easier to find information in a specific context.

After finding the results of the search, the CGI script dynamically creates a VoiceXML page to send back to the voice browser based on the results found. Special cases are handled if no results are found or if a long list of results is returned. The VoiceXML page that is returned also includes an option for returning to the starting VoiceXML page if the user wishes to make another query.

7. Conclusions and Future Work

We have developed a system that is a functional starting point for exploration of user interfaces that are able to build and utilize shared context with users. Using the WebContext system, users can remotely access information that was built while browsing web pages on a desktop computer. We have demonstrated a voice user interface for accessing and querying this shared context.

While building the WebContext system, we have identified several areas for future work:

- *Automated techniques for gathering additional context information*. Our current system only uses the content (text) of the web pages to build shared context. We would like to incorporate additional information such as page view duration, viewing history, and mouse activity into our context models.
- *More robust query capabilities*. This version of WebContext uses a very simple query model that only allows users to request one piece of information based on two context indicators. In addition to expanding the query capabilities, we are examining how to characterize applications in terms of the information types and context indicators needed.
- *Use of more robust language modeling.* As mentioned in section 5.2.3, speech recognition systems can be configured using a statistical language model created from an input corpus of text rather than from a context-free grammar (our current system uses a CFG grammar). The use of CFGs for WebContext presents some significant scalability problems. Specifically, using the simple grammar construction techniques presented here, the size of the grammar grows quickly with the number of pages viewed. We believe this recognition task is well suited to statistical language techniques and plan to incorporate a statistical language model into our next version of WebContext. We may also explore the use of weighted-CFGs in which the transitions in the grammar are each given a probability (or weight).
- *Heuristics about what to include in the grammar*. Even with statistical language modeling, we plan to explore techniques for deciding what terms and combinations of terms to include (or techniques for how to weight such terms) in the grammar/language model.
- *Automatic inclusion of synonyms and related concepts*. We anticipate that users will not always remember the exact words and phrases on web pages they viewed. For this reason, we would like to extend the terms in the language model to include words and phrases that are similar to or related to terms that are actually on the web pages. We have begun looking at the use of WordNet (http://www.cogsci.princeton.edu/~wn) for this purpose.

8. Acknowledgments

This research was supported by the National Science Foundation under grants IIS-9876167 and DGE-9553458. Seth Golub has put together a system called autojot for archiving web pages that helped form some of the concepts and architecture of WebContext.

9. References

[1] Adar, E., Karger, D., and Stein L. A.. Haystack: Per-User Information Environments. Proceedings of the Eighth International Conference on Information Knowledge Management, 413-422, Kansas City, MO, 1999.

[2] All Things Web (ATW). How Much Is Too Much? The State of the Web Survey. Conducted May, 1999. Web page accessed May 2, 2001: www.pantos.org/atw/35654.html

[3] Brown, M.K., Glinski, S.C., Goldman, B.P., and Schmult, B.C. PhoneBrowser: A Web-Content-Programmable Speech Processing Platform. The W3C Workshop on Voice Browsers, Cambridge, MA, 1998.

[4] Budzik, J., and Hammond, K.J. User Interactions with Everyday Applications as Context for Just-in-time Information Access. International Conference on Intelligent User Interfaces 2000, 44-51, New Orleans, LA, 2000.

[5] Chen, L., and Sycara, K. WebMate: A Personal Agent for Browsing and Searching. Proceedings of the 2nd International Conference on Autonomous Agents and Multi Agent Systems, Minneapolis, MN, 1998.

[6] Christian, K., Kules, B., Shneiderman, B., and Youssef, A. A Comparison of Voice Controlled and Mouse Controlled Web Browsing. ASSETS 2000, 72-79, Arlington, VA, 2000.

[7] Clark, H. *Arenas of Language Use*. Chicago: University of Chicago Press, 1992, p. 3.

[8] Cole, R., Mariani, J., Uszkoreit, H., Zaenen, A., and Zue, V. Survey of the State of the Art in Human Language Technology. Edited report, 1996. Web version accessed October 1, 2001: http://cslu.cse.ogi.edu/HLTsurvey/HLTsurvey.html

[9] Cyveillance, "Internet Exceeds 2 Billion Pages," Press Release, July 10, 2000. Web page accessed October 1, 2001: http://www.cyveillance.com/web/us/newsroom/releases/2000/2000-07-10.htm

[10] Hemphill, C. and Thrift, P. Surfing the Web by Voice. Proceedings of the Third ACM International Conference on Multimedia, 215-222, San Francisco, CA, 1995.

[11] Goecks, J., and Shavlik, J. Learning Users' Interests by Unobtrusively Observing Their Normal Behavior. Proceedings of the 2000 International Conference on Intelligent User Interfaces, 129-132, New Orleans, 2000.

[12] Joachims, T., Freitag, D., and Mitchell, T. WebWatcher: A Tour Guide for the World Wide Web. Proceedings of the International Joint Conference on Artificial Intelligence, 1997.

[13] Java Speech Grammar Format Specification Version 1.0, October 26, 1998. On-line document accessed on October 1, 2001: http://java.sun.com/products/java-media/speech/forDevelopers/JSGF.ps.

[14] S. Lawrence. Context in Web Search. IEEE Data Engineering Bulletin, vol. 23, pp. 25-32, 2000.

[15] H. Lieberman. Letizia: An Agent that Assists Web Browsing. Proceedings of the International Joint Conference on Artificial Intelligence 1995, Montreal, 1995.

[16] H. Lieberman, N. W. Van Dyke, and A. S. Vivacqua. Let's Browse: A Collaborative Web Browsing Agent. International Conference on Intelligent User Interfaces 1999, 65-68, Redondo Beach, CA, 1999.

[17] Marais, H., and Bharat, K. Supporting Cooperative and Personal Surfing with a Desktop Assistant. 10th Annual Symposium on User Interface Software and Technology, 129-138, Banff, Alberta, Canada, 1997.

[18] Marx, M. and Schmandt, C. Putting People First: Specifying Proper Names in Speech Interfaces. Proceedings of the ACM Symposium on User Interface Software and Technology, 29-37, 1994.

[19] McCarthy, J.C. and Monk, A.F. Channels, conversation, cooperation and relevance: all you wanted to know about communication but were afraid to ask. Collaborative Computing, 1, 35-60, 1994.

[20] Mladenic, D. Machine learning used by Personal WebWatcher. Proceedings of ACAI-99 Workshop on Machine Learning and Intelligent Agents, Chania, Crete, 1999.

[21] Robin, M., and Hemphill, C. Considerations in Producing a Commercial Voice Browser. The W3C Workshop on Voice Browsers, Cambridge, MA, 1998.

[22] Schmandt, C. Phoneshell: the Telephone as Computer Terminal. Proceedings of the First ACM International Conference on Multimedia, 373-382, Anaheim, CA, 1993.

[23] VoiceXML Forum, "Voice eXtensible Markup Language VoiceXML version 1.0." Specification, March 7, 2000. On-line specification accessed on October 1, 2001: http://www.voicexml.org/specs/VoiceXML-100.pdf

[24] World Wide Web Consortium. W3C Workshop on Voice Browsers, Cambridge, MA, 1998. Workshop web site accessed September 30, 2001: http://www.w3.org/Voice/1998/Workshop/

[25] Yankelovich, N., Levow, G.A., and Marx, M. Designing SpeechActs: Issues in Speech User Interfaces. Conference Proceedings on Human Factors in Computing Systems, 369-376, Denver, CO USA, 1995.