Personal Naming Environments

Margaret M. Fleck
Mobile and Media Systems Laboratory
HP Laboratories Palo Alto

HPL-2003-108
May 22nd , 2003*

E-mail: fleck@hpl.hp.com

naming, ubiquitous computing

Bindings from names to resources play a key role in ubiquitous computing systems. They can allow users to retrieve virtual resources by producing a natural language name or by scanning the ID on a physical object. Shared names could also be used to group related resources semi-automatically. To be maximally useful, sets of bindings (environments) must be personalized to a user and/or context. Busy users, however, have little time to devote to authoring. This paper presents a lightweight system for quickly authoring environments. Its new features include a simple mechanism for "mixing" existing environments into new creations, ability to repurpose legacy web content, and support for an interactive authoring technique that we call "authoring by playing."

Personal Naming Environments

Margaret M. Fleck

Hewlett-Packard Labs, 1501 Page Mill Rd, MS 1138
Palo Alto, CA 94304-1126, USA
`fleck@hpl.hp.com`,
`http://www.hpl.hp.com/personal/Margaret_Fleck`

Abstract. Bindings from names to resources play a key role in ubiquitous computing systems. They can allow users to retrieve virtual resources by producing a natural language name or by scanning the ID on a physical object. Shared names could also be used to group related resources semi-automatically. To be maximally useful, sets of bindings (environments) must be personalized to a user and/or context. Busy users, however, have little time to devote to authoring.
This paper presents a lightweight system for quickly authoring environments. Its new features include a simple mechanism for "mixing" existing environments into new creations, ability to repurpose legacy web content, and support for an interactive authoring technique that we call "authoring by playing."

1 Introduction

Computer users notoriously accumulate large quantities of digital resources (e.g. email messages, downloaded papers, bookmarks) much faster than they can organize them. The new trends in mobile devices simply make the problem worse, by giving users more ways to pick up items (e.g. digital cameras, handheld ID readers), more places to store them, and less powerful display technologies to use for access and management. A widespread but unrealized dream is that assets will somehow organize themselves. Failing that, we need simple ways for busy users to provide key organizational information.

This paper considers a limited part of this problem: managing the bindings that map names to resources. Name bindings play four key roles in ubiquitous interfaces. They provide links between related resources (i.e. standard hyperlinks). They let physical objects evoke on-line resources, by providing meanings for the arbitrary ID's embedded in computer-readable ID tags. They let users retrieve resources by speaking or writing memorable natural language names, more practical for portable devices than wading through long menus. Finally, shared names impose a flexible structure on resources, by implicitly grouping together resources that are probably related.

However, the full power of name bindings can only be realized when they are tuned to the user and his interests. Suppose that I ask a hallway display screen to show information on "Yann." The speech recognizer needs to have this unusual word in its vocabulary, so that it isn't confidently misrecognized as "yawn." Moreover, I want to see information for my friend Yann Le Cun, not all the other Yann's that Google happens to know about. [1] Thus, the application running on the display must have access to my personal selection of bindings, my personal *naming environment.*

The key challenge is to make naming environments easy to create dynamically. Personal bindings must be created and maintained for each user, making the most efficient use of the very limited effort they will put into authoring. Users may wish to create naming environments specific to a particular interest (e.g. hobby), facet of their life (e.g. work), or occasion (e.g. a formal presentation at a conference). Programs acting on the user's behalf might adapt his environment to his current context, e.g. physical location. Programs managing shared devices (e.g. wall displays) must retrieve and use bindings appropriate to the set of users who happen to be present.

This paper describes a system for quickly authoring naming environments. Its features include a simple mechanism for "mixing" existing environments into new creations, ability to repurpose legacy web content, and support for an authoring technique we call *authoring by playing.* It forms part of our research program on "ubimedia", i.e. hypermedia for the ubiquitous user [2].

2 A Scenario

To understand the uses and requirements for a naming system, imagine that Aaron and Xandie are visiting Oxford, England. As they prepare for their trip, they browse the internet using their laptops. They find several on-line tours of Oxford, e.g. one from the University Chemistry department, one from the City Council, and a theme tour on Tolkien's Oxford. Each tour has been written by a different person and covers a different selection of the landmarks. However, they all link standard place names such as "Merton College" to information about those places. These shared names allow the three tours to be automatically combined ("mixed") into one tour that is more comprehensive.

While Aaron and Xandie tour Oxford, they carry only small handhelds. To retrieve information about a landmark (which might or might not be nearby), they might speak its name or use graffiti. The naming environment of the pre-loaded tours provides a preferred vocabulary that helps the speech or handwriting recognizers understand their inputs. The handheld then displays resources from the selected tours.

While mobile, the travellers also acquire new resources. They might "bookmark" objects seen at the Pitt Rivers Museum by scanning ID tags on the display cases. They may take photographs, binding them to location names (e.g. "Botanic Garden") via speech input or GPS location. These names and bindings to resources are all stored in their personal historical records.

When they return home, they may tell and re-tell the story of their trip, to different audiences. The couple might first sit down at a shared display to look

[1] Unless, of course, I explicitly request a search.

through their bookmarks and photos. The interface should automatically combine their name bindings. Thus, saying "Merton" will play all the photos that either person took there and saying "shrunken heads" [2] will play the appropriate museum caption, no matter which of them actually bookmarked that case. Xandie's historical record now contains all the items she viewed with Aaron.

Before she goes to visit her mother, Xandie might review her historical record again, selecting and playing only items that would appeal to her mother. She extracts this piece of her record as a separate file named 4mom. When she plays 4mom at her mother's house, the shared display combines her name bindings with her mother's. So saying "Merton" now displays not only Xandie and Aaron's photos, but also a photo of her mother taken at Merton. This new binding for "Merton" is captured in Xandie's historical record, so she can look back at it later.

Thus, Xandie's record of her trip changes each time she tells the story. Viewing it in the company of others brings in new bindings and their associated resources, enlarging the record. Selective replaying and extraction of subsections changes the order in which landmarks are displayed and refines the set of resources associated with each one. We call this process *authoring by playing.* It is similar to how people refine oral narratives (e.g. stories, songs). It is also similar to the fast prototyping technique available in interpreted computer languages such as LISP: try out ideas one by one in an interpreter, then cut and paste the good parts into program files.

Not everything in a ubiquitous computing interface can be accomplished via name bindings. The goal of this paper is to squeeze the maximum possible value out of a relatively simple name binding system. This will help clarify what additional mechanisms might actually be required.

3 Names, Environments, and Narratives

The above scenario depends on the key notions of context-sensitive names, ubiquitous hyperlinks, naming environments, and narrative records.

3.1 Context-sensitive Names

Most of the names we encounter in our daily lives are context-sensitive, i.e. evaluated with respect to some designated naming environment. For example, the intended referent of "Northampton" depends on whether I live in the US or the UK. The function denoted by "cosine" adapts automatically to what mathematical computing packages I have loaded. And the memory location bound to the variable "PageCounter" changes as my executable is swapped in and out of main memory. Standard hyperlinks are anomalous in being relatively closely bound to a specific storage location.

Previous authors [9, 12] have noted several advantages of replacing the URLs in hyperlinks by context-sensitive names. By changing the naming environment

[2] A well-known exhibit at the Pitt Rivers.

in which its hyperlinks are evaluated, we can adapt documents to the current interests and backgrounds of the users present. The document automatically picks up any new name bindings added by the user (e.g. a new photograph bound to the name "Gene") and less editing is required when URLs change. A single hyperlink can point to a group of related web resources rather than just a single one. Finally, because names need not be globally unique, they can be kept short enough for users to easily type or speak.

Names can take a variety of forms. This paper uses natural language names for readability and to emphasize natural language interfaces. However, some identifiers might be opaque ID's such as UPC codes, MD5 summaries (e.g. of JPEG images), or those embedded in physical tags (e.g. RFID tags). Some names might be globally unique identifiers (e.g. [10]) and some unique only when enough context is supplied.

In order for a naming system to be useful, authors and users must agree on how to name objects of interest. Many sorts of names have been well-standardized by social mechanisms, e.g. proper names of people and places ("Tim Kindberg"), names of many common objects ("eggplant" or "nutmeg") technical jargon ("passiflora" or "abelian group"), serial and model numbers of manufactured goods, and ID's embedded in tags to be physically attached to objects. Modest amounts of variation in names (e.g. "Timothy" vs. "Tim") can be handled successfully using standard techniques from natural language understanding. However, this sort of naming system is not intended to handle freeform descriptive noun phrases such as "Fredrick's tie-dyed potholder."

3.2 Ubiquitous Hyperlinks

In physical user interfaces (e.g. [11, 12, 16, 20, 23]), the physical world becomes a (3D) document which can be browsed in much the same way as an HTML document. Tagged physical objects are viewed as *physical hyperlinks*[12]. Tapping or scanning a physical object then yields web resources related to that object. Similarly, language-based interfaces allow the user to retrieve web resources related to an object he sees or remembers, by producing (e.g. speaking, typing) a natural language name for the object (e.g. "JFK Airport"). Again, we can view these *natural language hyperlinks* as basically similar to standard HTML hyperlinks.

Given an appropriate naming environment, all three types of hyperlinks (standard, physical, language) can be dereferenced in much the same way to yield web resources. By gluing together the physical and virtual worlds, physical and natural language hyperlinks make communicating with the computer more similar to one's interactions with people and non-electronic devices.

3.3 Bindings and Environments

A *naming environment* is an ordered list of *name bindings.* The most basic possible representation of a name binding is a pair (N, R), where N is a string name and R is the URL of a resource. Resolving a name N in an environment E yields an ordered list of all bindings in E whose name matches N.

This definition of name resolution differs slightly from standard practice in (say) programming language design, where resolution yields a unique result. As the scenario in Section 2 illustrates, we welcome multiple return values. However, we do expect only a reasonable and modest number of return values, not the very long lists of vaguely relevant items returned by search engines such as Google.

In pervasive computing applications, we expect to find errors and inconsistencies in names, because these are typically input (directly or indirectly) by the users. Therefore, we must make allowances for such errors when deciding whether an input name N matches the name in a binding B. Our current prototype requires that N be a substring of B's name, ignoring case distinctions. A smarter algorithm might incorporate some knowledge of natural language (e.g. understanding common inflectional affixes) and/or common user interface errors (e.g. words that sound similar to a speech recognizer).

The current implementation returns bindings in the order in which they appear in the environment, eliminating duplications. The end of the list containing the putatively most salient bindings will be called the "front". The output order should eventually also be influenced by how well each binding's name matches the input string N, i.e. approximate matches appear later than exact matches.

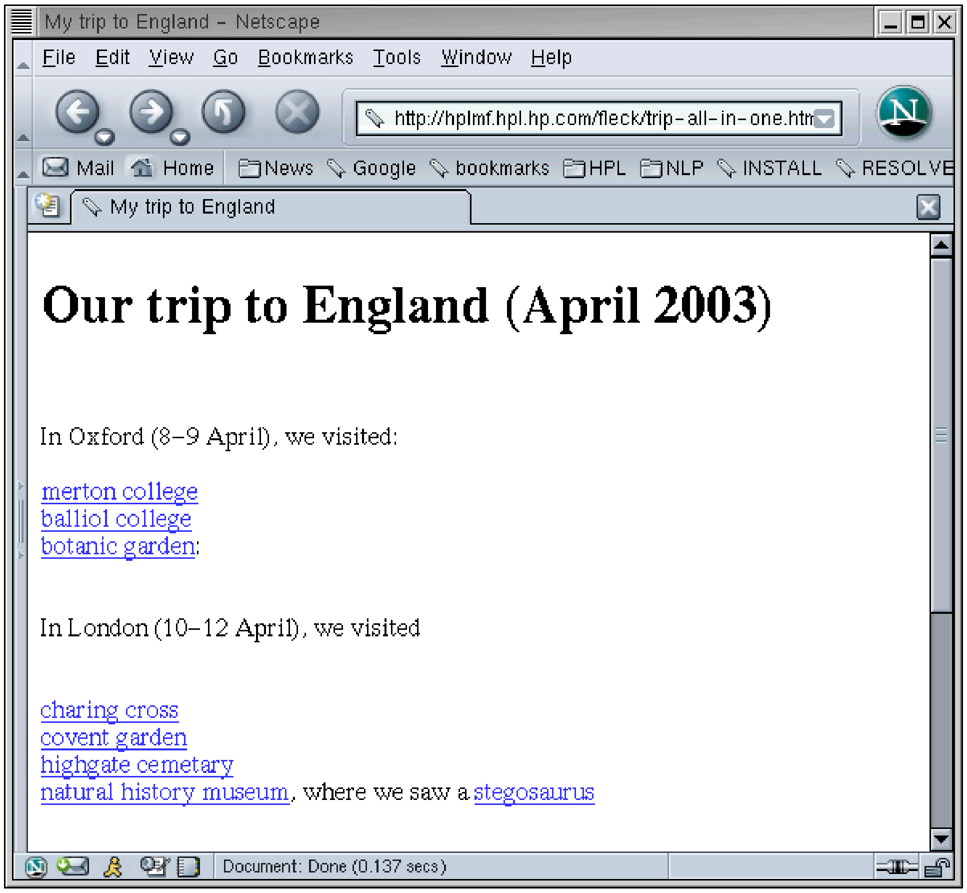
3.4 Naming vs. Searching

Naming environments are not intended to replace search engines or search interfaces to large collections (e.g. all books sold by Amazon.com). The primary job of search engines (e.g. Google) is to help users find resources they haven't seen before. Or to return to a resource when they have lost or failed to capture the key facts about that resource. A naming environment allows the user to quickly return to a familiar resource that he has recorded as useful. Naming environments are best seen as a smarter replacement for browser bookmarking.

Moreover, search engines are context-independent. The results for a given query depend only on that query. They usually do not depend on the user (personal search engines e.g. [8] are an exception) and never depend on his current context or interests. Choosing a naming environment allows the user to impose a particular viewpoint on the set of available resources, which restricts the set of results returned for each query. For example, "Alex" refers to one of two colleagues when I'm at work, but to my nephew when I'm talking to my parents.

3.5 Narratives

The users in our scenario create documents that incorporate context-sensitive hyperlinks. The initial seed documents are automatically-generated *narrative records*, chronological records of the user's actions as seen by an on-line recording service, e.g. as in Figure 1. Our previous experiments at the San Francisco Exploratorium [6, 7] show that such bare-bones records can be acquired easily and are useful to visitors.


My trip to England - Netscape
File Edit View Go Bookmarks Tools Window Help
http://hplmf.hpl.hp.com/fleck/trip-all-in-one.htm
Mail Home News Google bookmarks HPL NLP INSTALL RESOLVE
My trip to England
Document: Done (0.137 secs)

Our trip to England (April 2003)

In Oxford (8–9 April), we visited:

merton college
balliol college
botanic garden;

In London (10–12 April), we visited

charing cross
covent garden
highgate cemetary
natural history museum, where we saw a stegosaurus

Fig. 1. A very basic "seed" narrative record.

These seed documents allow the user to select and replay some of his actions at a later time. The narrative records created in these replaying sessions gradually become *prompt sheets* that help the user remember the flow of which items he likes to retrieve. Finally, selected prompt sheets might be edited into a more pretty and polished form, e.g. similar to a typical web page, for distribution to friends and relatives.

4 Personal Naming Environments

A full demonstration of name-based interfaces for computing system would require three sorts of components: a back-end service to manage naming environments (the "personal mixer"), client applications that use the mixer, and applications that distribute environments and narratives to others. This paper will concentrate on the design of the mixer. The mixer's behavior will be illustrated using a simple browser-based client application, with discussion of how one might design the variety of clients required for mobile users. Finally, we will assume the standard existing distribution mechanisms, especially email, web servers, and global search engines.

The personal mixer must support (at least) three types of client user interfaces. Busy mobile users will perform only very basic actions, e.g. load a set of bindings (e.g. guidebook supplied at the entrance to a museum), play information associated with a name, and bookmark a physical object. Users sitting quietly for a few minutes might browse their historical record, search the web for more resources, and the like. Finally, they will occasionally sit down at their own personal home interface to edit items for distribution to family and friends.

To do this, the mixer API must include five types of capabilities:

- basic operations: create a binding, resolve a name in the working environment
- bulk operations: install a whole set of bindings into the working environment, clear the working environment
- browse the historical record, replay selected prior actions
- extract parts of the historical record for publication as playable narrative records and/or installable environment files
- export information required by "smart" user interface programs, e.g. vocabulary information needed for speech recognition

Prior researchers have demonstrated basic bind/resolve functionality. Therefore, we will focus on the more advanced features required for historical replay and authoring, concentrating on creating tools that are simple, lightweight, and facilitate re-use of existing materials. Because no one has extensive experience authoring naming environments for pervasive computing, we will borrow techniques that have established themselves as successful in authoring code and text documents.

Specifically, we will use a declarative publication model, i.e. distribute naming environments as text-like documents. This allows client applications to use a global search engine to discover useful environments authored by strangers, e.g. environments binding some opaque ID they have found on an object.

Historical records will be displayed in the same format used for environments that are to be installed. This enables "copycoding": cut-and-paste copying from old creations, including the historical record, into new creations, either verbatim or with subsequent editing. The system allows *inclusion by reference*: documents defining environments can include the content of an old creation in a new creation via a command that names the location of the old creation (like the #include command in C).

Finally, the order of bindings in the environment is determined implicitly by the order in which bindings are created and environment files are loaded. It's easy to imagine more sophisticated methods. However, load order is simple, easy to understand, minimizes authoring effort for the user (no "magic numbers"), and works reasonably well. It is not obvious which other methods would actually improve on it.

```
BIND google http://www.google.com
INSTALL http://sample.my.org/myveggieenv
INSTALL http://sample.seeds.com/catalog/vegetables.html
BIND eggplant http://sample.gardens.com/catalog/vegetables/eggplant.html
BIND chives http://sample.gardens.com/catalog/herbs/chives.html
```

Fig. 2. Idealized environment file

5 Prototype System

The current prototype system consists of four component services: the mixer, the UI assistant, the summarizer, and the stub service. The mixer is the basic back-end service which maintains the user's personal environment. The UI assistant and stub help a standard browser pretend to be a next-generation browser that understands context-sensitive names. The summarizer provides capsule summaries of web resources, required to help the UI assistant do its job.

5.1 Personal Environment Mixer

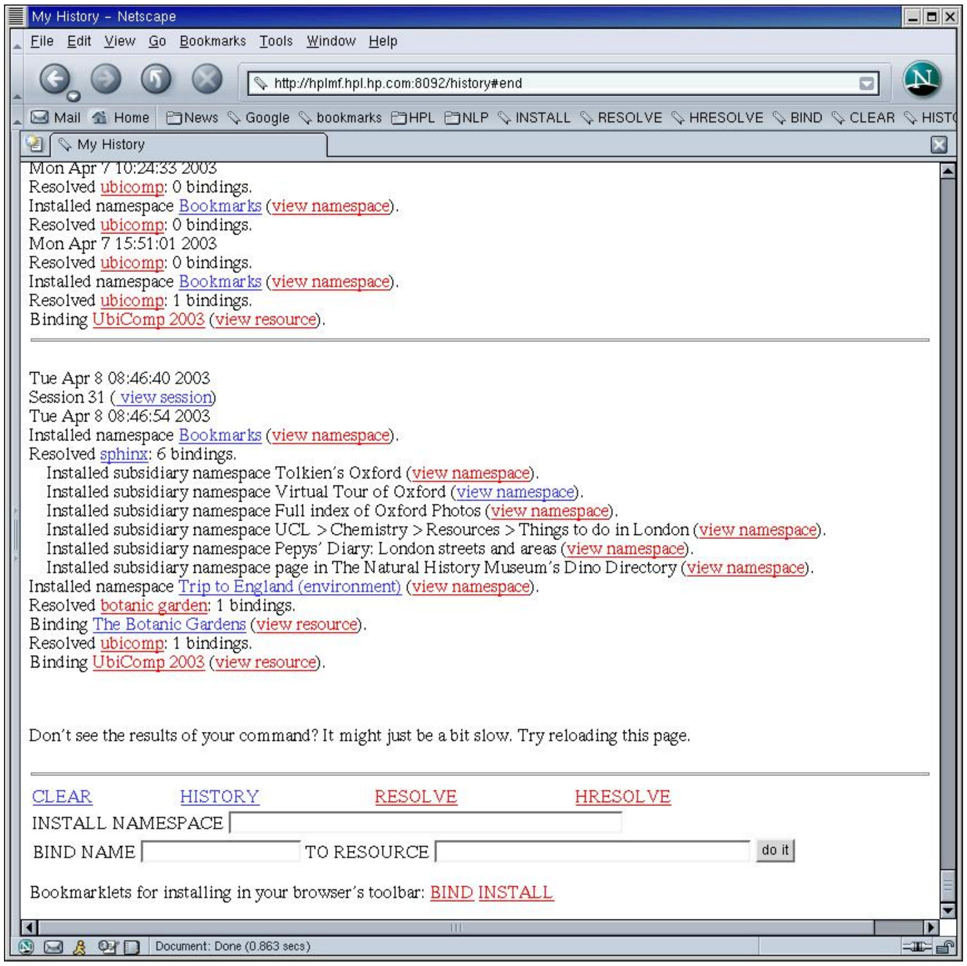
The personal environment mixer maintains the user's current naming environment. In its most basic form (see Section 7 for more features), the mixer accepts HTTP requests to bind a name to a resource, install a file of bindings into the environment, resolve a name against the current environment, and clear the environment.

Figure 2 shows an idealized environment file, abstracting away from all issues of format. It is essentially a scripting file, containing an ordered list of commands. When the mixer is told to install this file, it executes the commands in order. Each BIND command causes the (name, resource) pair to be pushed onto the front of the environment. Each INSTALL command causes the named file to be recursively loaded and installed. In other words, environment files are loaded exactly like files of declarations in most programming languages.

Notice that the environment contains the bindings in the inverse of the order in which they were received. When we resolve a name N, the bindings matching N are returned in the same order as they appear in the environment. That is, the most recently loaded binding will appear first in the resolution results. Issuing a new BIND request for a binding already in the environment causes that binding to migrate to the front of the environment and, thus, appear at the top of the resolution results.

5.2 The Browser Interface

The mixer can be accessed via a simple browser-based UI. Although the interface is deskbound, it uses extremely simple interaction patterns. The interface is a browser (e.g. Netscape, IE) augmented with a set of buttons (a mixture


My History - Netscape
File Edit View Go Bookmarks Tools Window Help
http://hplmf.hpl.hp.com:8092/history#end
Mail Home News Google bookmarks HPL NLP INSTALL RESOLVE HRESOLVE BIND CLEAR HISTO
My History

Mon Apr 7 10:24:33 2003
Resolved ubicomp: 0 bindings.
Installed namespace Bookmarks (view namespace).
Resolved ubicomp: 0 bindings.
Mon Apr 7 15:51:01 2003
Resolved ubicomp: 0 bindings.
Installed namespace Bookmarks (view namespace).
Resolved ubicomp: 1 bindings.
Binding UbiComp 2003 (view resource).

Tue Apr 8 08:46:40 2003
Session 31 (view session)
Tue Apr 8 08:46:54 2003
Installed namespace Bookmarks (view namespace).
Resolved sphinx: 6 bindings.
  Installed subsidiary namespace Tolkien's Oxford (view namespace).
  Installed subsidiary namespace Virtual Tour of Oxford (view namespace).
  Installed subsidiary namespace Full index of Oxford Photos (view namespace).
  Installed subsidiary namespace UCL > Chemistry > Resources > Things to do in London (view namespace).
  Installed subsidiary namespace Pepys' Diary: London streets and areas (view namespace).
  Installed subsidiary namespace page in The Natural History Museum's Dino Directory (view namespace).
Installed namespace Trip to England (environment) (view namespace).
Resolved botanic garden: 1 bindings.
Binding The Botanic Gardens (view resource).
Resolved ubicomp: 1 bindings.
Binding UbiComp 2003 (view resource).

Don't see the results of your command? It might just be a bit slow. Try reloading this page.

CLEAR HISTORY RESOLVE HRESOLVE
INSTALL NAMESPACE
BIND NAME TO RESOURCE do it

Bookmarklets for installing in your browser's toolbar: BIND INSTALL

Fig. 3. A sample narrative historical record. Horizontal bars indicate when the environment was cleared.

of bookmarks and Javascript bookmarklets). The user presses the CLEAR button to clear the environment. To install bindings from a web page P into his environment, the user navigates to P, then clicks the INSTALL button.

After most commands, the browser displays the user's naming history (Figure 3). The browser is pointed at the last entry in the history, which will show the results of the most recent request. For slow commands such as installation of complex environments, progress reports will appear gradually at the end of the history. The historical display produces a quick feedback loop similar to that of a LISP interpreter.

The RESOLVE button pops up a dialog box to obtain the name to resolve, then displays the resolution results. Similarly, the BIND button obtains a string from the user, then binds this to the browser's current page. Experiments with loading Netscape bookmark files into the environment (see below) strongly suggest adding a BOOKMARK operation that binds the current page to its title.

A mobile handheld device would require two main adaptations. First, the dialog box for input of names would be replaced with input from speech recognition, ID scanners, and the like. Second, some of the web pages might be displayed via audio rather than graphically.

5.3 The UI Assistant

Only some of these user interface operations can be supported with the raw interface to the mixer, which is primarily designed to be friendly to programs. The UI assistant mediates between the user's browser and the mixer, restructuring the mixer's outputs to make up for the limitations of the legacy browser. For example, the raw output of most mixer commands is an HTTP 200 acknowledgement of success, suitable for a wide range of client applications but somewhat laconic. It is the UI assistant that redirects the client's browser to display the user's naming history for more complete information.

When the user asks to resolve a name N, the mixer returns a bare list of binding pairs. The assistant decides how to present this list to the user. The current prototype uses a very simple strategy. If resolution yields only one binding and that resource is responsive, the browser is redirected to it. Otherwise, the assistant creates a selection menu, similar to the output of a search engine (e.g. Google).

There are many other possible ways to present a list of bindings. We might display only the first resource, by itself or with a concise selection menu under it (e.g. in another HTML frame). We might display the top several resources in parallel, either on multiple display screens (if available), by splitting the browser screen into several frames, or by playing audio files while displaying visual ones. We might even merge the contents of the resources, e.g. using a multi-document summarizer to produce a joint summary of several text documents.

In the long term, the best choice of display strategy depends on the user's preferences, on the types of resources in the list, and on the display devices currently available to him. For example, the user interface might attempt to use all available display screens to display as many results as possible in parallel.

5.4 Displaying Selection Menus

An apparent problem with the basic binding model is that the raw list of bindings contains only URLs and short names. This isn't sufficient information about the resources to let the user easily identify which one he wanted. I address this problem by invoking a "summarizer" service, which takes a URI as input and returns a capsule summary of the resource found when that URI is dereferenced. The UI assistant asks the summarizer for information about each resource and then produces a selection page similar to that of a web browser.

We anticipate that the summarization service might be used by a variety of applications, not just those related to personal naming environments. Therefore, the service provides not only a text summary of the resource, but also (as available) its MIME type, title, and MD5 digest. It might also include size in bytes, thumbnails of images, and/or estimate of network delay required to retrieve the resource. The text summaries are currently computed by converting the resource to plaintext (if possible) and extracting the first 200 words. The service caches information and updates it when convenient. Like a search engine, it does not attempt to track all versions of fast-changing web resources (e.g. news sites).

5.5 Monitoring Usage

When the user issues a RESOLVE or HRESOLVE request, the UI assistant detects which bindings were actually dereferenced, via specially encoded URLs in the selection menus. For each such binding, it generates a BIND request to the mixer. This makes the binding migrate to the front of the environment, so that it will appear at the top of the selection list next time. As we will see in Section 7, this binding is also logged in the user's historical record and migrates to the front of his historical environment.

Pushing bindings to the front of the environment when they are used is an extremely simple way to re-order resolution results based on the user's behavior. It is easy to imagine more complicated systems (e.g. monitor how often a resource is accessed). However, the simple method seems to be fairly effective and it is unclear which upgrades would actually improve performance.

5.6 Context-Sensitive Hyperlinks

Embedding context-sensitive names in HTML (or similar) documents requires working around the limitations of current browsers. Ideally, when the user clicks on a name-based hyperlink, the name should be dispatched to the user's personal mixer for resolving. The resulting resource or resources should then be displayed in the browser.

Eventually, next-generation browsers might recognize a special URL prefix (e.g. "env:") for names that need to be resolved. To emulate this effect with a standard browser, we can install a special service on the user's machine at port (say) 48888. Hyperlinks created with the prefix http://127.0.0.1:48888 will then be dispatched to this service. [3] For example, a context-sensitive hyperlink for the name "Alex" would look like:

```
http://127.0.0.1:48888/resolve?name=alex
```

In principle, the service at port 48888 could be the UI assistant. To avoid short-term hassles porting code to different operating systems, I have instead installed a very simple "stub" service at the magic port. The stub service redirects

[3] It is necessary to use the numerical address because Netscape interprets references to localhost incorrectly.

the request to his personal UI assistant, which can live anywhere on the network. Only the stub service needs to be portable across operating systems. This design also makes it possible to quickly switch between two running versions of the UI assistant and/or mixer, e.g. for demos and testing.

Notice that we now have, for free, a mechanism for binding one name to another: bind the first name to a context-sensitive hyperlink containing the second name. This is one method by which users could inform the naming system that two names are related or (by linking in both directions) synonymous.

6 HTML Environment Authoring

So far, this paper has been deliberately vague about the format used to encode lists of bindings. We could invent some special purpose markup language to distribute sets of name bindings, e.g. some dialect of XML as in [12, 25]. However, this would require persuading large numbers of people to adopt the new format, including necessary code or reauthoring of legacy content. In a world where reputable commercial sites can't necessarily put titles in their html markup, this seems likely to take a long while.

Notice, however, that HTML pages frequently contain hyperlinks whose anchor text names the resource at the end of the hyperlink. For example, the following link can be viewed as associating the name "University of Iowa" with a specific web resource.

```
<a href="http://www.uiowa.edu"> University of Iowa</a>
```

Let's call these "naming-type hyperlinks." The popularity of such links has been exploited by a number of recent search engines [3, 4, 17], which use anchor texts as a primary technique for associating possible query terms with web pages.

There are many web pages that consist largely of naming-type hyperlinks, embedded in modest amounts of explanatory text. These include on-line catalogs, hobby and fan sites, tourist guides, "people" listings and portal pages for various organizations (e.g. http://www.w3c.org). The three Oxford tours described in Section 2 actually exist (as of the date this paper was written) and have this format. Netscape stores its bookmarks in this format. Such pages can be seen as legacy naming environments.

Inspired by this source of legacy content, the prototype naming system described in this paper uses HTML as its format for publication of environments. Specifically, when the mixer is asked to install a page of HTML, it interprets each standard hyperlink in the web page as a BIND command. However, hyperlinks whose resource has the magic prefix described in Section 5.6, which I will call "command-type URLs", are interpreted instead as commands to the environment mixer. Our prototype environment pages currently use two command-type URLs:

http://127.0.0.1:48888/install?resource=XXX is a command to recursively install resource XXX.

http://127.0.0.1:48888/view?resource=XXX causes this hyperlink not to be interpreted as a BIND command.

In other words, HTML is being used as a primitive scripting language.

This HTML-based format for environments is far from ideal. (Some might call it a perverse kludge.) However, it is approximately as compact and readable as comparable XML formats and it offers significant short-term advantages: it is familiar to the community and our applications can make use of existing, mature tools such as browsers, HTML editors, and email. This approach could help us determine what features are (and are not) important in an ideal long-term format and could co-exist with a better solution as it is being phased in.

Because it can parse HTML, the prototype can repurpose legacy web content, as well as new content created by people who haven't yet bought into our system. This lets the system offer an immediate benefit to early adopters. (Installing one's Netscape bookmarks file is a good way to get started.) Using existing content, we can quickly construct large sets of test and demo data which reflect the actual interests of real web users. This will help us accurately predict the eventual usage pattern.

Finally, a single web page can serve as both the machine-readable definition of an environment and a human-friendly description of it, i.e. be *dual-use*. Many existing pages of links (e.g. the Oxford tours) contain text and graphics at the top and/or bottom describing the authorship, theme, and/or intended audience. Comments on the quality of resources (e.g. "no longer maintained") may be found next to hyperlinks. HTML markup can be used to indicate internal logical structure (as is done in Netscape bookmark files). The VIEW command-type URL can be used to direct the human reader to related content without adding these links to the set of bindings.

The dual-use feature allows the point-and-click installation protocol from Section 5.2 to be implemented with a trivial bookmarklet and use legacy web pages. A similar feature used by Kindberg [12], required special links in the pages. In Section 7 we will see how dual-use helps in displaying historical information. Finally, a document can serve as its own environment. For example, the story of a wedding might start with narrative text, using hyperlinks containing names like "John". At the end comes a list of hyperlinks binding these names to resources, rather like a bibliography at the end of a research paper. The reader can then navigate to the document, push a browser button to install it, then browse it like an ordinary HTML page.

7 Authoring by Playing

The personal environment mixer also maintains a record of the user's naming-related actions, called his *personal naming history*. The history consists of two components: a narrative record and a historical environment. Three commands are added to the interface for the personal environment mixer: HISTORY, SESSION, and HRESOLVE. The resulting historical interface is somewhat minimal, but supports a simple version of authoring by playing.

The narrative record (Figure 3) returned by the HISTORY command is modelled after the standard Emacs interface to a LISP interpreter. It shows all key actions that the user has taken, e.g. all environment files installed, all names resolved, and all bindings "used" (see below). It is divided into numbered sessions that can be retrieved individually using the SESSION command. A new session is started whenever the environment is cleared.

The narrative record is also used to inform the user about any errors encountered in executing these commands, e.g. unavailable environment files. Timestamps are shown when the environment is cleared and where there is a significant lapse of time between successive actions. Entries in the narrative record must be kept concise, otherwise it becomes unreadable.

The narrative record is exported as HTML. Its hyperlinks are designed so that clicking on the record of an action causes that action to be repeated. This allows one to quickly repeat a sequence of actions, with selective omissions and deletions. Moreover, if a session S is extracted into a new file F, installing F will rebuild all the bindings created or loaded during S. The VIEW command-type URL is used to let the user see (rather than install) a previously-installed environment file.

The personal naming history also keeps an environment containing "historical" bindings. This is a focused list of items that the user has expressed an actual interest in, not the full list of all bindings embedded somewhere in environments he has installed. More recently-used items are listed in front of older ones. The mixer leaves the decision of what belongs in this list to the client application. The HRESOLVE command resolves an input name against this list.

In the current prototype, the UI assistant asks the mixer to record in the historical environment all BIND requests issued at top-level. This includes BIND commands issued directly by the user and also those issued by the mixer when the user dereferences a resolution result.

In the current prototype user interface, RESOLVE and HRESOLVE are separate commands. Other user interfaces might choose to have only one user-level resolve command, returning the combined results of both underlying operations. The results might be presented as one unified list of resources. Or they might be presented in separate frames.

The combination of the narrative record and the historical environment allow the user to experiment with various combinations of bindings and environments. The results from a successful experiment can be extracted via the SESSION command, loaded into your favorite HTML editor, given some minimal editing (e.g. a title), and then published on the web. It is easy to revisit what you have seen in the past, by reviewing your historical record or resolving names historically. This is the basic essence of authoring by playing.

8 Status and Next Steps

The author has been using the prototype implementation for several months, as a replacement for the grossly inadequate bookmark management facilities of standard browsers. Despite known limitations mentioned above, it is easy to use, effective for finding previously-seen web pages, and works well with moderately large sets of bindings. For example, my environment currently contains 2144 bindings, of which 601 were imported from my Netscape bookmarks. Convincing testing would, of course, require more users and much larger sets of bindings.

8.1 More Client Applications

The prototype's most pressing need is for a wider range of client applications. We hope first to connect it to a speech interface, so that resources can be retrieved by speaking, rather than typing, their names. This will test the system's ability to handle errorful input and test the hypothesis that we can achieve acceptable speech recognition in a ubiquitous computing environment by providing well-targeted vocabulary. It is also critical for usability in hands-busy applications such as providing information in the kitchen.

To port the prototype's interface to shared screens in public places ("digital furniture"), we hope to upgrade the UI assistant to use information about which users are present (e.g. users might identify themselves to the system with RFID cards). Requests made at the display (e.g. resolving a name) must be transmitted to mixers for all the users, because it may be difficult to identify which user has made the request and so that each user's history gets a complete record of what he has viewed.

We also hope to experiment with different ways to display multiple resolution results and "play" documents containing context-sensitive hyperlinks. For example, if two displays are available, should they be used to display the top two resolution results, or the top result and a selection menu? Which display shows the top result? Should historical bindings be displayed automatically? Should they be mixed in with current bindings or separate? Should results from several users (at a shared display) be mixed together or displayed separately?

Finally, other members of our research group are designing client applications for small handhelds, which can capture data from ID tags and ship it to selected services, using lightweight point-and-click user interfaces. Others are working on techniques to adapt display of data semi-automatically to which display devices (display screens, speakers, etc) are available [2].

8.2 Errorful Inputs

The prototype uses an excessively primitive method for matching names. The matching should be done on a word-by-word basis, do proper approximate string matching, and include primitive linguistic analysis e.g. a stemmer [22]. The name-matching mechanism might also need to understand the internal structure of dates and GPS locations.

Combining the mixer with a speech front-end is not a simple matter of transcribing the speech into a string which is then passed to the mixer to resolve. Current speech recognizers make many errors when attempting to transcribe words not in their precompiled vocabularies [15]. The mixer does not, and should not, contain specific information about low-level phonetics and the acoustic properties of the room and microphone being used. Therefore, cooperation between the two components will require careful design. Similar issues apply to interfaces based on handwriting recognition, completion of partial typed inputs, and the like.

8.3 Web Issues

Fetching environment files from the web can take significant amounts of time, as can fetching web resources that need to be summarized. There are interesting issues to be explored involving how much the system should cache, how often it should attempt to update its fetched data, and whether it can pre-fetch certain resources (e.g. resources mentioned in an environment that has just been loaded).

A significant number of web sites organize their naming-type hyperlinks into a hierarchical tree of web pages. Smart crawling software would be required to extract an entire such tree, so that it can be installed as a unit into one's naming environment. Other miscellaneous issues include how to use resources that require authentication and how to synchronize several distributed copies of one's personal naming environment.

9 Related Work

This paper extends the scope of work by Hartman et al. [9] and Kindberg [12]. Hartman et al. introduced the basic concepts of context-sensitive links, but name resolution is incorporated into the browser and naming environments are associated with documents rather than users. Kindberg developed a similar mechanism as a free-standing service, but presents it only in the context of relatively well-managed sets of names and resolving physical hyperlinks. Similar representations are used in RSS [25], but for describing links rather than creating name bindings.

These previous systems were not designed for interactive authoring and each contains only some of the key features required to do this well. Only Kindberg's system supports interactive use and the interactive API is incomplete (e.g. no clear or bulk upload operations). Only Hartman allows one environment file to include another by reference. [4] Only RSS contains any support for repurposing legacy documents. None can display a record of the user's prior actions. All (especially RSS) add extra fields to the basic representation of bindings without compelling arguments for their usefulness.

In particular, previous systems include a descriptive title in each binding, despite the fact that a summarization service can provide adequate information automatically. For example, the Google search engine summarizes the textual parts of world-readable web pages. Our text summarizer is very basic. Mani [19] reviews the extensive prior work on better summarization techniques.

[4] Kindberg's technique of dispatching to a peer resolver is a good solution to a different problem: ad hoc combination of a handful of personal environments. It does not scale well for creating complex environments from many inputs.

Finally, browser bookmarking and history facilities capture web resources together with their titles and some limited other information (e.g. date). The result is effectively a list of bindings and, as we saw above, Netscape bookmark files can be used as such. However, users must manage their bookmark collections using very primitive interfaces, which scale poorly to large collections. User studies [1, 24, 18]. show that this is a significant annoyance and suggest a variety of possible improvements, including better searching and history facilities.

A number of groups have experimented with systems to index files viewed in the browser[8, 20], personal notes or events captured by the mobile user[5, 13, 14], personal photographs[21], and voicemail [26, 27]. There seems to be evidence users might like to retrieve such personal media items using a combination of full-text search (on text files or transcribed speech), automatically-gathered contextual information (e.g. time, location, title), and annotations explicitly added by the user. However, it is still difficult to draw strong conclusions, in part due to the limitations of existing systems for speech recognition and context capture.

10 Conclusions

Short of presenting extensive user studies or large-scale deployments (very rare), authors rarely convince their readers that they have created the best and simplest user interface. Rather, I hope I have convinced you of four points:

- Context-sensitive names might be useful in semi-automatic organization of digital assets for a mobile user.
- Authoring naming enviroments requires good tools, not just a lookup table.
- Simple mechanisms (e.g. for ordering bindings) can be quite powerful. More complex methods require thoughtful justification.
- Repurposing legacy content might be important for doing tests of meaningful size in the short term.

References

1. David Abrams, Ron Baecker, and Mark Chignell (1998) "Information Archiving with Bookmarks: Personal Web Space Construction and Organization," CHI 1998.
2. Barton, John; Goddi, Patrick; Spasojevic, Mirjana (2003) "Creating and Experiencing Ubimedia", Hewlett-Packard Labs, Tech. Report HPL-2003-38.
3. Sergey Brin and Lawrence Page (1998) "The Anatomy of a Large-Scale Hypertextual Web Search Engine," Proc. Seventh Int. Conf. on the World Wide Web.
4. Soumen Chakrabarti et al. (1999) "Mining the Link Structure of the World Wide Web," IEEE *Computer* 32/8, pp. 60-67.
5. I. B. Crabtree and B. J. Rhodes (1998) "Wearable computing and the remembrance agent", BT Techn. Journ. 16/3, pp. 118–124.
6. Margaret Fleck, Marcos Frid, Tim Kindberg, Eamonn O'Brien-Strain, Rakhi Rajani, Mirjana Spasojevic (2002) "From Informing to Remembering: Deploying a Ubiquitous System in an Interactive Science Museum" IEEE *Pervasive Computing*, pp. 13-21.

7. Margaret Fleck, Marcos Frid, Tim Kindberg, Eamonn O'Brien-Strain, Rakhi Rajani, Mirjana Spasojevic (2002) "Rememberer: A Tool for Capturing Museum Visits," Proc. UbiComp 2002.
8. Seth Golub, "Autojot", http://www.aigeek.com/geek/autojot/.
9. John H. Hartman, Todd A Proebsting, Rajesh Sundaram (1997) "Index-Based Hyperlinks," Computer Networks and ISDN Systems 29, pp. 1129-1135.
10. Sandro Hawke, "Tag URI," http://taguri.org.
11. Hiroshi Ishii and Brygg Ullmer (1997) "Tangible Bits: Towards Seamless Interfaces between People, Bits and Atoms," Proc. CHI 1997, pp. 234–241.
12. Tim Kindberg (2002) "Implementing physical hyperlinks using ubiquitous identifier resolution", Eleventh Intern. Conf. on World Wide Web 2002, pp. 191-199.
13. Mik Lamming and Mike Flynn (1994) "'Forget-me-not': Intimate Computing in Support of Human Memory", FRIEND21 Symposium on Next Generation Human Interface.
14. Mik Lamming et al. (1994) "The Design of a Human Memory Prosthesis", *The Computer Journal*, 37/3, pp. 153–163.
15. Beth Logan, Pedro Moreno, Om Deshmukh (2002) "Word and Sub-word Indexing Approaches for Reducing the Effects of OOV Queries on Spoken Audio," Human Language Technology 2002, pp. 31–35.
16. Peter Ljungstrand, Johan Redström and Lars Erik Holmquist (2000) "WebStickers: Using Physical Tokens to Access, Manage and Share Bookmarks to the Web, " Proc. Designing Augmented Reality Environments 2000.
17. Oliver McBryan (1994) "GENVL and WWWW: Tools for Taming the Web," First Int. Conf. on the World Wide Web.
18. Bruce McKenzie and Andy Cockburn (2001) "An Empirical Analysis of Web Page Revisitation", 34th Hawaiian Intern. Conf. on System Sciences.
19. Inderjeet Mani (2001) "Automatic Summarization," John Benjamins, Amsterdam.
20. Hannes Marais and Krishna Bharat (1997) "Supporting Cooperative and Personal Surfing with a Destop Assistant," ACM UIST 1997, pp. 129–138.
21. Timothy Mills, David Pye, David Sinclair, and Kenneth Wood (2000) "Shoebox: A Digital Photo Management System," TR 2000-10, AT&T Labs, Cambridge, UK.
22. M. F. Porter (1980) "An algorithm for suffix stripping," *Program*, 14(3), pp. 130-137.
23. Jun Rekimoto and Yuji Ayatsuka (2000) "CyberCode: Designing Augmented Reality Environments with Visual Tags", Proc. Designing Augmented Reality Environments 2000.
24. Abigail Sellen, Rachel Murphy, Kate Shaw (2002) "How Knowledge Workers Use the Web," CHI 2002, pp. 227-234.
25. Dave Winer, "RSS 2.0," http://backend.userland.com/rss.
26. David Whittaker, Richard Davis, Julia Hirschberg, Urs Muller (2000) "Jotmail: a voicemail interface that emables you to see what was said", CHI 2000, pp. 89-96.
27. David Whittaker et al. (2002) "SCANMail: a voicmail interface that makes speech browsable, readable, and searchable," CHI 2002, pp. 275–282.