The FirstSearch User Interface Architecture:
Universal Access for any User, in many Languages, on any Platform

Gary Perlman
OCLC Online Computer Library Center, Inc.
6565 Frantz Road
Dublin, Ohio 43107 USA
+1 614 761 5058
perlman@acm.org

ABSTRACT

The OCLC FirstSearch® service allows users to search for bibliographic and full text records in over 80 online databases. Web-based, FirstSearch was designed to adapt to unexpected user needs, platform considerations, languages, and changing requirements. The many unknowns during development necessitated an architecture that would allow many types of contributors to modify the interface easily and frequently. For example, marketing, documentation, and user interface designers edited the strings used in the interface, including translation; and user interface and graphic designers edited the screen layout. Structured initialization files with a simple convention for adapting to specific users, platforms, languages, etc., allowed continual broadening of the accessibility of the system without complicating the overall architecture.

The paper begins with a discussion of the general requirements for FirstSearch (multi-platform, multi-lingual, levels of users, low-end hardware, accessible) and the need for better coordination of contributions from the FirstSearch team. The architecture is then described, which partitions the specification of the interface into platform - specific, language-specific, and language/platform independent functional components. The user interface, in the form of Web pages, is then generated dynamically (although it would also be possible to generate static pages). The paper ends with a discussion of experiences with the changes to the interface and a cost-benefit analysis of the architecture, with the overall conclusion that addressing many accessibility issues in the architecture facilitated individual accessibility issues.

Keywords

Accessibility; customization; declarative interface specification; disabilities; disabled; diversity; global; globalization; globalisation; group and individual differences; handicapped; i18n; impaired; impairment; intercultural; international; internationalization; multilingual; software localization; special needs; translation; universal access;

> *"The power of the Web is in its universality. Access by everyone regardless of disability is an essential aspect."*
> Tim Berners-Lee, http://www.w3.org/WAI/

PROBLEM: SECOND SYSTEM EFFECT

The OCLC FirstSearch® service is a Web-based bibliographic and full text retrieval system (at NewFirstSearch.oclc.org). FirstSearch delivers to libraries and their patrons over 80 databases, each with about 10-30 indexes (e.g., keyword, author, title, subject, date, ...), to access a combined total of over 200 million records and millions of full text articles. Over 15,000 libraries subscribe to the service, submitting at times over 300,000 searches per day.

Originally delivered as a text-based system in 1991, the Web-based version of FirstSearch, released in 1996, steadily gained popularity over the text version. The system was built on the OCLC SiteSearch® Z39.50 Web-server, which maintains persistent information about a user's search/retrieval session. The original SiteSearch was developed in C and contained several embedded proprietary languages developed by the OCLC Office of Research. The desire to add new features and to be able to adapt more easily to user needs motivated the development of a new version of FirstSearch, built on the new Java-based version of SiteSearch. The new FirstSearch would have many new features:

- limit results to a library's holdings (books / serials)
- limit results to full text available to an account
- search across multiple databases, made possible in part by standardizing indexing across all databases
- sorting and ranking options
- wildcards and truncation
- integrated thesaurus
- a high degree of customization

The difficulty of adding desired features in the old system, and the relative liberation of a dynamic Java environment, led to what Brooks (1975, Ch.5) might call a *second-system effect* of trying to incorporate every feature for which there was a desire in the first system.

About a year was spent developing the detailed requirements, during which time design options were unclear, because we were in areas of little experience:

- new functionality
- new application layer and search engine (SiteSearch, being developed during and after FirstSearch requirements)
- new programming language (Java)
- new version of operating system on new hardware, including a new *high-performance* file system

There were several general requirements for the user interface of new FirstSearch. Although we had devoted considerable effort to the detailed requirements, cross-platform, text-only, multilingual, and accessibility and help requirements were expressed with little more than a sentence each.

Multi-Platform: Run on Everything

The system would work on all *current* browsers. We proposed supporting the 4.x versions of

- Netscape Navigator (used by about half our users)
- Microsoft Internet Explorer

Initially, we wanted to require JavaScript, and even CSS (Cascading Style Sheets), but Navigator 4.0 had limited support for features we could provide by other means. Then, we realized that many sites (e.g. libraries with hundreds of old machines) would not be willing or able to upgrade their browsers. We committed to support the 3.x versions of Navigator and Explorer, although with some compromises due to limited functionality (e.g., MSIE 3). We committed to support:

- JavaScript enabled
- JavaScript disabled, missing, or lacking

We committed to support different screen sizes:

- large: more than 900 pixels wide
- medium: from 700 to 900 pixels wide (most common among our users)
- small: less than 700 pixels wide

across different hardware/operating system:

- Windows
- Macintosh

and to test 256-color screens and for grayscale contrast.

Text-only Version: Run on any Hardware

In addition to the graphical UI browsers, we wanted to support a version that could run in a telnet window because, for a small number of high-frequency users, telnet is the only access. We chose to create a version of the interface that would work reasonably well with Lynx, a text-based HTML browser that would run on our server. This would replace the telnet text version of FirstSearch.

Multilingual Interface: Run in Many Languages

We planned to translate the interface and online help into three languages initially:

- English
- French
- Spanish

We hoped this would be easier than our multilingual effort in the old Web-based version (Hysell & Perlman, 1999) which was not developed with internationalization in mind.

Universally Accessible: Run for Everyone

There was a requirement to be "ADA compliant" (Americans with Disabilities Act) although with no knowledge of what that entailed, not even that, at the time, there were no defined standards. Initially, we thought that the text-only Lynx interface would serve that purpose, but later found that text-screen-readers serve some users, while specially adapted graphical browsers serve others.

Levels of Users: Usable and Useful to Everyone

We planned to support three search modes:

- Basic, with support for the most used features
- Advanced, with all features
- Expert, to better support query language users

Library and Patron Customization

We planned to allow libraries to customize their version of FirstSearch, setting default options, including branding elements such as library logo. We planned to consider allowing individual patrons to set personal preferences that would persist across sessions.

Group Coordination Issues

Different groups of people were given primary responsibility for the different dimensions of the user interface:

- **marketing**, for requirements and terminology
- **development**, for functionality
- **database**, for loading new databases
- **graphic design**, for icons, fonts, colors, and layout
- **usability**, for interaction design/re-design
- **documentation (including translation)**, for on-screen help, online/printed help

This is an over-simplification because many groups contributed to many dimensions. Still, few individuals think about all the above concerns when working on a specific task, so it was a user interface coordination goal to make sure that, for example:

- developers did not put any non-portable platform-specific HTML or language-specific terms in Java code or database configuration files
- graphic or user interface design that worked on one platform worked on all platforms, particularly if it used JavaScript
- terminology that was used in one part of the system would be used consistently in all parts of the system (including help) and that the terminology physically fit in the space allocated (in all languages)

DESIGN APPROACH: CROSS-PRODUCTS OF PARTITIONED INFORMATION TO DEFER DECISIONS

FirstSearch required a generalized approach to ensuring universal access because there multiple dimensions of accessibility: platform, language, disability, etc. (Perlman, 1999). The highest priority goal for the user interface architecture was to be able to **adapt the interface to inevitable requests for changes** (due to as yet unknown usability, performance, functional, etc. considerations). Because there were so many unknowns, the **architecture had to be incrementally scaleable** starting with a simple model of the user interface, but able to expand, as we understood more. Previous experience (Perlman, 1989) suggested that partitioning the system into orthogonal sets of information and building the system by forming the cross-product of those sets would allow incremental elaboration and optimal redesign. The method is similar to word-processing mail merge except that instead of inserting address information into letter/label templates, attributes of functions are inserted into platform-specific templates for Web pages. The partitioning chosen included:

- **functional aspects**: specific functions for database selection, search, and results
- **platform-dependent aspects**: adapting the display to different platforms
- **language-dependent aspects**: language used in the system, gathered together for easier translation

Functional Partition: The FSPage Model

The first step in developing the UI architecture for FirstSearch was to apply information design to identify some parts (attributes) of *pages*. A page is an object with the information used to construct what a user observes and does on a single Web page. Initially, the specific *pages* in the system were unimportant because we knew that new pages would be added and some existing pages would be merged with others or deleted. A canonical sequence of pages in a FirstSearch session is database-selection, search, and results. Similar to pages, the specific *attributes* of pages were unimportant because it should be easy to add/delete/change attributes. Each page in FirstSearch has:

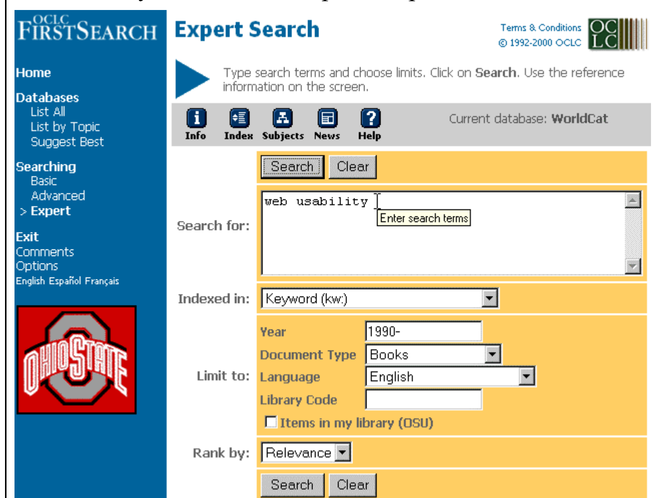
- **pagename**: an internal identifier
- **pagetitle**: a title displayed to users
- **pagelabel**: a short phrase for links in menus
- **tips**: on-screen help tips
- **status**: on-screen status information
- **controls**: page-specific controls
- **action**: a form action
- **panel**: a main form panel

Individual pages can have any of about 10 other specialized attributes (e.g., how to process form elements in the panel, error handling). Attributes can contain constant text and any number of *entities* (SiteSearch constants, variables, and Java method calls), so they are highly dynamic.

To make these pages platform-independent and language-independent, we extracted the platform-dependent parts into a style file and the language-dependent parts into a language file. We replaced what was extracted with *entities* defined in configuration files (called INI files). The resulting platform- and language-independent page definitions were placed in `pages.ini`, a configuration file with a section for each page. For example, the definition for the expert search page looks something like **Specification 1**, which appears to users like **Screen 1.**

Specification 1: Expert search page.

```
[expert]
pagename      = expert
pagetitle     = &Lang.pagetitle.expert;
pagelabel     = &Lang.pagelabel.expert;
tips          = &Lang.tips.expert;
status        = &Lang.status.expert;
controls      =
   &Style.dbinfo.gadget;
   &Style.scanindex.gadget;
   &Style.thesaurus.gadget;
   &Style.news.gadget;
action        = /FSQUERY:searchtype=expert
term          = termexpert
index         = indexexpert
focus         = termexpert
panel         =
   &Style.dialog.begin;
      &Pages.basic.submit;
      &Pages.expert.searchbox;
      &Pages.expert.index;
      &Pages.advanced.limits;
      &Pages.advanced.options;
      &Pages.basic.submit;
   &Style.dialog.end;
searchbox     =
   &Style.dialog.rowbegin;
      &Style.font.labelbegin;
         <label for=termexpert>
            &Lang.label.find;
         </label>
      &Style.font.labelend;
      &Style.dialog.elementbegin;
        <textarea name=termexpert id=termexpert>
            &termexpert;
        </textarea>
      &Style.dialog.elementend;
   &Style.dialog.rowend;
```



Screen 1: Expert search screen rendered on Explorer 5. Accessibility features show *tips* on input elements.

Many of the attributes are references to language entities (e.g., `&Lang.tips.expert;`) defined in language files (one for each language). Some of the attributes are style entities, used to mark the beginning and end of structurally meaningful parts (e.g., `&Style.dialog.begin/end;`). Other entities include references to other parts of pages, so that definitions can be modular and reused (e.g., all search screens use the Basic **submit** buttons defined in `&Pages.basic.submit;`). Special purpose attributes indicate the names of terms and indexes used on search screens, and where to focus the cursor if JavaScript is available. **Specification 2** shows a different page.

```
Specification 2: Detailed record page.
[record]
pagename     = record
pagetitle    = &Lang.pagetitle.record;
pagelabel    = &Lang.pagelabel.record;
tips         = &Lang.tips.record;
status       = &Lang.status.record;
controls     =
   &Style.thesaurus.gadget;
   &Style.ill.gadget;
   &Style.holdings.gadget;
   &Style.email.gadget;
   &Style.print.gadget;
action       = /FSFETCH:fetchtype=record
panel        =
   &Style.dialog.begin;
      &Style.dbsuggest.gadget;
      &Style.navigate.gadget;
      &Style.record.gadget;
      &Style.navigate.gadget;
   &Style.dialog.end;
```

Practical Partitioning

With the page object defined, the most difficult aspect of partitioning the language and style information was locating the information to be partitioned and then doing it consistently. We found that it was nearly impossible to explain to developers why and how to keep this information separate, perhaps because it required an understanding of translation, cross-platform development, accessibility issues, and general usability considerations. Another problem with having developers create language-independent and platform-independent code was that the language and the formats were being developed as the system was being built.

Our approach was to have developers create screens using *untamed* English and HTML and then partition the information for them. Platform-specific parts were extracted and replaced with style entities for runtime substitution based on the user's platform and preferences. See **Figure 1**.To internationalize the design, we moved language strings into a language file, replacing them with language entities to create language-independent HTML; later, the language-specific values would be inserted into the HTML by substituting language entity values in the user's language. See **Figure 2.** During the process, choice of language and interface design could be reviewed, and later, design decisions were centralized so that decisions could be changed.

Figure 1: Partitioning platform-dependent information into a style file for later dynamic entity substitution.

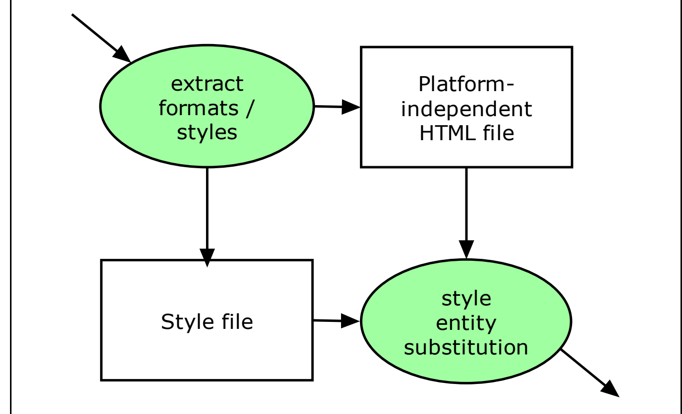
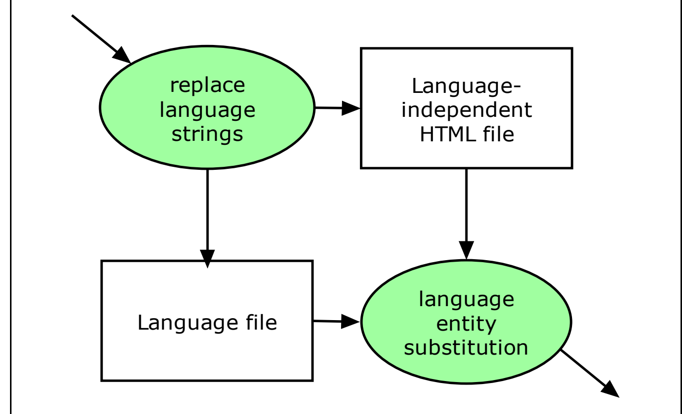


Figure 2: Partitioning language strings into a language file for later dynamic entity substitution.



Platform-Dependent Partition

The FSPage object contains references to platform-dependent parts, parts that will display differently on different platforms, and which will require different HTML. There are many factors that may affect how the interface design is presented to users, including:

- browser name and version (and sub-version)
- operating system
- screen size
- whether JavaScript is enabled

For example, if JavaScript is available, a Help window can pop up, be sized based on the screen dimensions, and show shortcut keys that are based on the operating system.

There are many methods to adapt to different displays, ranging from using *lowest-common denominator* features, to unique sub-sites for different displays, to dynamically customized displays. Given the number of platforms planned for FirstSearch, and the many differences among these platforms, dynamic generation of HTML was an obvious choice.

There are many ways to implement dynamic generation of HTML. At an architectural level, these should be interchangeable to adapt to changing technology.

- **XML:** was untried, and many of the features we needed were already provided by SiteSearch
- **CSS:** SiteSearch entity substitution provided the features we needed, and for non-CSS browsers.
- **Java applets:** Java was unacceptable to many users because of security / performance concerns.

The method chosen for FirstSearch was designed to abstractly represent the structure of displays separate from the final rendering. For example, an *untamed* error message might be initially marked up as:

```
<font color=red size=5><b>
   Something bad happened
</b></font>
```

Styles could be replaced by entities:

```
&ErrorBegin;
   Something bad happened
&ErrorEnd;
```

and defined elsewhere:

```
[styles]
ErrorBegin = <font color=red size=5><b>
ErrorEnd   = </b></font>
```

A line in a search form might be marked up as:

```
&SearchFormBegin;
   ...
   &SearchLineBegin;
      &LabelBegin;
         Find:
      &LabelEnd;
      &FormElementBegin;
         <input type=text name=terms>
      &FormElementEnd;
   &SearchLineEnd;
   ...
&SearchFormEnd;
```

These examples have been simplified to better explain the methods used in FirstSearch; the real versions have many gory details. SiteSearch, on which FirstSearch is built, allows the definition of *entities* that are substituted into the outgoing HTML. So by changing the definition of these structural entities, we can change the HTML that will be generated. For example:

- on graphical browsers: the error message above might be preceded by an error icon and appear in large red font
  - If JavaScript is enabled: the error message might be placed in an alert box (although it is not in FirstSearch)
  - On large screens: the message may be made 2 sizes larger
  - On medium screens: the message may be made 1 size larger
  - On small screens: the message may be left the same size
- on a non-graphical browser (e.g., Lynx): the error message may be bold and surrounded by lines

The fine granularity of control and the likelihood of editorial changes made it undesirable to code these changes in Java. Instead, a declarative method of specifying custom values was adopted.

1. When a session starts, all the potential customizing variables, e.g., browser attributes, are stored in about 30 entities.
2. Default entities (about 50) are read from an initializing configuration file.
3. Customizing entities are set based on values read from conditional INI-file sections.

Ordinary INI files contain named sections (e.g., **[styles]**) and entity definitions in those sections. *Conditional sections* are named by entity-value pairs. For example, the **browser** entity may be *Mozilla*, *MSIE*, *Lynx* or some other value. Conditional styles could be defined for each browser value, or for those that require special settings:

```
[styles]
section*   = browser
[browser=Lynx]
ErrorBegin = <hl><b>
ErrorEnd   = </b></hl>
[browser]
ErrorBegin = <font color=red size=5><b>
ErrorEnd   = </b></font>
```

The reference to `section*` causes the system to read the conditional section named `browser`. If `browser` is *Lynx*, the section called `[browser=Lynx]` is used. Otherwise, the default `[browser]` section is used. This can be further elaborated based on sets of conditional sections. For example, if we wanted the error message font size to depend on the screen size, we could insert an entity into the error message style and set the value of the entity in conditional sections:

```
[styles]
section*   = browser
section*   = screensize
[browser=Lynx]
ErrorBegin = <hl><b>
ErrorEnd   = </b></hl>
[browser]
ErrorBegin = <font color=red size=&ErrorSize;><b>
ErrorEnd   = </b></font>
[screensize=large]
ErrorSize  = 5
[screensize=medium]
ErrorSize  = 4
[screensize]
ErrorSize  = 3
```

Once conditional sections are set up, it is easy to add conditional entities. A major advantage of setting these in INI files is that the changes can be viewed while the system is running. The INI files can be re-read and entities re-set without changing any code. Another advantage is that the all the peculiarities of particular platforms are specified together. For example, the entire color scheme for MSIE 3 is different than for the rest of the system because that browser version does not support changing the color of text if it is in a hot link. Another example is that only MSIE 4+ handles Greek entities like `α`. for browsers that do not display them properly, we remove the entity delimiters (i.e., show "alpha" instead of α). FirstSearch uses conditional sections based on browser, operating system, screen size, JavaScript, and others to set over 100 entities.

Language Partition: Internationalization / Localization

FirstSearch was internationalized by moving all language-specific terms (about 5000) into INI files, and by replacing those terms by entities that refer to the section and entity name in that section. In FirstSearch language files, sections are used to distinguish how / where some text will be used. For example, all diagnostic message are stored in a section called `[msg]`:

```
[msg]
bad    = Something bad happened
nohits = Your search matched no records
nojs   = Your browser doesn't support JavaScript
```

Entities in language files are accessed by naming the entity (`Lang`) followed by the section (`msg`) and the variable name (`bad`) like this: `&Lang.msg.bad;`. So the platform-independent, language-independent version of the error message above becomes:

```
&ErrorBegin;
    &Lang.msg.bad;
&ErrorEnd;
```

When a user chooses a different language, entities in a different language INI file are associated with their session.

Structure: For the FSPage object, **pagetitle**, **tips**, and **status** are all sections in the language file. Each section contains variables, one for each page, defining the page title, on-screen help tips, and status. Being in the same section makes it easier to make the text for different screens consistent, both in English and when translating. It requires, however, that developers place page attributes in different sections of different INI files. To make it easier for the user interface and database groups to work together, we separated the user interface language INI file from a language file for database-specific terminology (which accounted for about two-thirds of the language used in the system). This reduced contention while both files went through hundreds of revisions.

English as the Second Language: Although the development was in English, with English strings being moved into the English language files for later translation, there was an initial *translation* step that took almost as long as the translation into Spanish and French. The initial language was a dialect of English used by librarians and developers of systems for searching library materials; call it *Jargonese*. Some of the terminology was inappropriate for library-naive users, all the more common because of the advent of the Web. So internally, a screen might be called "history", but to the users, it would be known as "Previous Searches", and we would actually display the language entity `&Lang.pagetitle.history;` so that if the name changed, the change would be propagated throughout the entire system, including documentation.

Finding Entities and Previewing Translation: To help translators and documentation writers determine where an entity was defined, we created an *entity language* in which the value of an entity was the section and variable name where it was defined (e.g., `&Lang.pagetitle.history;` would be displayed as `pagetitle~history`). Then they would know that the string displayed in English as "Previous Searches" was the `history` variable in the `pagetitle` section. To help translators see their translations, we provided a facility for dynamic reloading of entity values on the current screen so they would not need to start a new session. Preview was important because the translators needed to ensure that their edits fit and did not break any embedded HTML or entities. Marketing used preview to review English that was replacing *Jargonese*.

Screen 2: French version of the expert screen. Users can change language at the bottom of the left navigation menu.

Template-Based Page Generation

The first prototype systems were based on *ad hoc* format flat-file databases accessed with **perl** scripts to generate HTML files. Eventually, the information about pages migrated into semi-structured INI files, and HTML files evolved into dynamically-generated HTML. The pages and attributes evolved over time, gradually increasing in complexity, and keeping them in an easily editable format was a positive feature.

HTML pages are generated in FirstSearch by inserting page-specific entities into templates (See **Specification 3**). FirstSearch templates were created for the graphical-browser version, the text-only Lynx version, a printable (cleaned up) version, etc. Templates can (and perhaps should) start as simple renderings of some attributes, but are scaleable in that they can be augmented easily. Templates can also facilitate major changes. Initial versions of the FirstSearch interface were framed, but because of transaction costs, we decided to evaluate an unframed version. Creating an unframed version of the whole interface took about an hour, and the results motivated us to change to an unframed interface. This change was made by one person, changing one set of templates into a single template, in less than a day. At various times, we have been able to design, create, and view completely different interface designs that were fully functional systems.

Specification 3 is a simplified template for the Lynx interface, which is simpler than the graphical version. Note that most page attributes have been assigned to entities (e.g., the pagetitle is in `&FSpagetitle;`). The user's view of the screen is shown in **Screen 3**.

```
Specification 3: Template for Lynx displays
<html pagename="&FSpagename;">
<head>
   <title>&FSpagetitle;</title>
</head>
<body>
   &FSpagetips;
   &FSpagestatus;
   <form method="POST" action="&FSpageaction;">
      &FSpagepanel;
      &FSpagecontrols;
      &StyleTable.FSMenu.gadget;
   </form>
</body>
</html>
```

Screen 3: Expert screen for Lynx text-browser users. The navigation menu (not shown) is appended to the display.

```
                           Expert Search
Current database: WorldCat

Type search terms and choose limits.
Click on Search.

[Search]
dog_____________________________________
________________________________________
________________________________________
________________________________________
________________________________________
________________________________________
Indexed in: [Keyword (kw:)_________]
Limit to:
 Year          1990-______
 Document Type [Books______]
 Language      [English__________]
 Library Code  ___________
 [_] Items in my library (OCL)
Rank by: [No ranking__________]
[Search]

[info] [index] [subjects] [news] [help]
```

This Lynx template is one of many possible renderings of the parts of pages. One advantage of the framed version of the interface was that the main frame contained all and only the information that users would want to print. When the framing was removed, it took about an hour to create a template for a printer-friendly format that did not show menus and controls.

Templates are easy to evolve. To move the controls for all pages, move one line. To duplicate the controls above and below the main panel, copy one line. Many changes are unanticipated, so the flexibility of being able to make global changes is highly desirable. One developer wanted to place an entity value on every page in the system; it was a one-line change. Quality assurance wanted to add specially formatted comments to delimit logical sections of the screens (to help highlight differences in regression test scripts); the change took less time an hour.

Because templates can be defined hierarchically, they can share reusable parts. This can minimize the cost of new versions of templates, say, for a version that takes full advantage of Cascading Style Sheets.

Accessibility Issues

Initially, we thought that the text-only Lynx version would be the best platform for a screen-reader for a sight-impaired user. After interviewing one of our staff, who is blind and uses Web-aware HTML-reading software, we broadened our approach to include all browsers.

Because the HTML for formatting the display is localized in style files, most changes to adapt to the WAI Guidelines could be added centrally. Microsoft's Internet Explorer 4+ (MSIE4+) provides substantial support for accessibility-oriented tags, including some features useful for all users:

- **title**: The title attribute provides extra information about what it is attached to. FirstSearch uses title tags for input fields to provide more detailed prompts, and on links to explain where they will lead the user. See **Screen 1, 2, 3** text areas.
- **label**: The label tag allows a label to be more formally associated with a form element with which it is logically associated. Web screen readers know that a label is associated with a checkbox, and MSIE4+ lets users control form elements by clicking on their labels. See **Specification 1**.
- **accesskey**: The accesskey attribute allows Alt-x keys to be associated with form elements. FirstSearch associates Alt-s with submit buttons, and Alt-c with the clear button.

Levels of Users

FirstSearch is designed for different levels of users with three search levels: basic, advanced, and expert. These differ in the number of search boxes, number of indexes offered, number of limits shown, and the help offered.

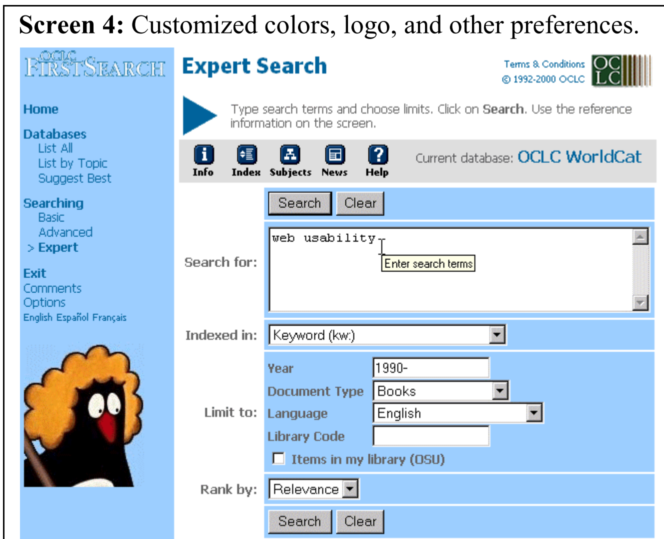
	Basic	**Advanced**	**Expert**
Search box	1 small	3 small	1 large
Indexes	3	10-15	20-30
Limits	full-text library	all	all
Help	simple examples	examples of more features	on-screen reference material

Customization

The FirstSearch administrative module allows libraries to customize FirstSearch: choosing default search modes, topic areas, library logo, links into library catalogs, and most options for controlling the access to for-fee items (e.g., full text of journals).

We are also exploring patron customization of the interface by saving settings across sessions. See **Screen 4**. Patron

settings were implemented in a few hours because the user interface architecture is designed to allow setting groups of entities. The same architecture is flexible enough for us to explore gender and age-based customization.



Screen 4: Customized colors, logo, and other preferences.

Coordination Issues

The partitioning of the user interface, and the plain text format of the interface initialization files, allowed non-developers to make changes to the developing system without involving programmers. For the first time, non-programmers had interactive control over the parts of the system for which they had responsibility, and it took many user interface decisions out of the hands of programmers (which was generally received positively by all).

Most contributors were not able to follow detailed instructions about how to develop platform-independent and language-independent screens. A few guidelines proved to be more effective (e.g., no HTML in Java code). For practical purposes, most programmers found it easier to write untamed code and partition it when it was ready. As problems were identified, checking scripts were enhanced to find problems automatically.

OBSERVATIONS AND CONCLUSIONS

The partitioned user interface architecture was designed with the main goal of being able to adapt to changing requirements. In achieving that goal, it allowed the rapid exploration and implementation of a variety of universal usability dimensions: cross-platform, multi-lingual, accessible, and in general, environment-sensitive versions. We could develop, largely with conditional sections of initialization files, parameters to adapt to the presence of JavaScript, quirky performance of certain browsers, custom parameters for different screen sizes, etc. The performance costs for dynamic generation of HTML have been small, and in some ways have improved performance because generated pages do not require any file access.

As we have gathered feedback and done more usability testing, we have made changes to the system. Returning to the highest priority goal of adaptability, it was not critical to get the design right, but it was critical to be able to change what was wrong. Global changes to reorganize all screens took minutes or hours of editing a single template instead of days or longer. Small changes have had invariably low costs, and larger changes have had generally proportional costs (although they sometimes have *multiplicative* benefits when applied to templates because templates apply to many pages).

What is perhaps the most striking result of designing an architecture capable of adapting to change is how it helped with areas for which we did not anticipate change. We knew the screen layout would change, and we knew the terminology would change, but we did not know we would be using label tags and title attributes for accessibility, nor that the same methods for adaptation to platform could be used for user-customizable versions of the system. The demands of a few universal access issues required a framework that helped address many.

REFERENCES

1. Brooks, F.P. *The Mythical Man-month: Essays on Software Engineering*. Addison-Wesley, 1975.
2. Hysell, D. & Perlman, G. *Lessons Learned from Internationalizing a Global Resource*, in G. Prabhu & E. delGaldo (Eds.) *Designing for Global Markets*, 1999, 183-192.
3. Perlman, G. "Coordinating Consistency of User Interfaces, Code, Online Help, and Documentation" in J. Nielsen (Ed.) *Coordinating User Interfaces for Consistency*, pp. 35-55, 1989, Academic Press.
4. Perlman, G. "CHI 99 SIG: Universal Web Access: Delivering Services to Everyone." *SIGCHI Bulletin,* 1999, 31:4, 53-54. Companion site at: http://www.acm.org/~perlman/access/

ACKNOWLEDGEMENT

I would like to thank Mike Prasse for his comments.