Porting Interactive Applications to the Web

Brian R. Gaines
Knowledge Science Institute
University of Calgary, Alberta, Canada T2N 1N4
gaines@cpsc.ucalgary.ca, http://ksi.cpsc.ucalgary.ca/KSI

Objectives of Tutorial

This tutorial is targeted on developers porting interactive applications from personal computers to operate in a client-server mode on World Wide Web (the *web*).The tutorial will be useful to those involved in porting, those responsible for estimating and managing porting effort, and those investigating the feasibility of porting applications to the web. A background in software development and basic knowledge of HTML/HTTP are assumed.

1 Porting Applications to Operate in Client-Server Mode on the Web

Many interactive applications have been written to operate effectively on personal computers without access to networks. With the growth of access to networks there is often user interest in having these applications able to take advantage of a networked environment and:-

- access data on other systems
- make data available to others
- integrate with other applications
- run on a range of platforms
- operate as groupware with multiple collaborating users.

Figure 1 shows the client-server architecture of the web. A client accesses servers on the Internet using various protocols. It communicates with various helper applications that extend its functionality. When it accesses a World Wide Web server using the HTTP protocol that server can also access various helper applications through server gateways.

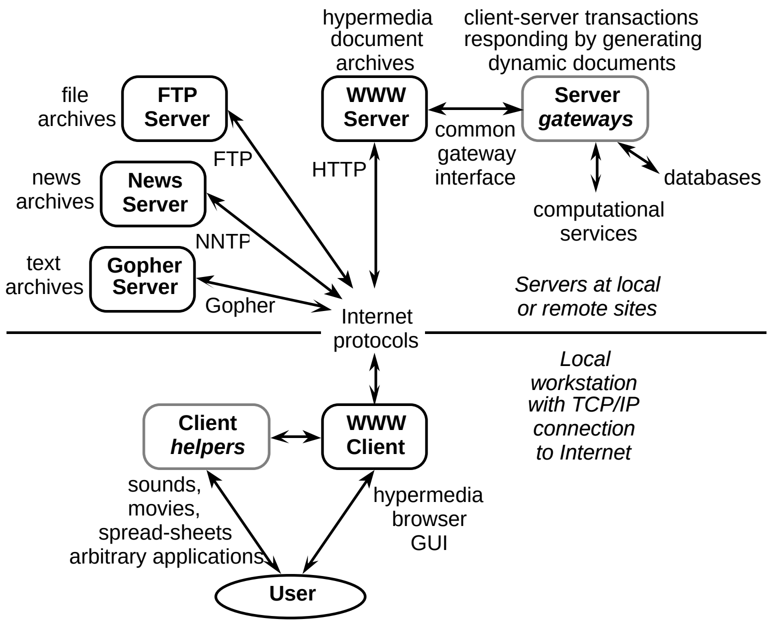


Figure 1 Client-server architecture of World Wide Web

It is not simple to convert an application designed for individual use to operate effectively in a client-server environment, particularly if cross-platform operation is also required. If one forgets the groupware problems of access to shared data structures initially, the problem reduces to one of factoring the application into two components:-

- an application client
  - running locally
  - interfaced to user through local graphic user interface (GUI)
  - interfaced to server through network
  - usually cross-platform implementation
- an application server
  - running remotely
  - no user interface
  - interfaced to client through network
  - possibly cross-platform implementation.

The server implementation requirements are easier to satisfy than those for the client because no user interface is required. Even if the server has to operate on a number of platforms this is substantially easier if only computational and network code has to be ported.

Web client implementation requirements are such that if a special-purpose client has to be written then little may be gained through factoring the application. The removal of the actual application functionality may gain little, and the requirements to operate through the network and be available on multiple platforms may add substantially to the task of implementing the client.

However, if a standard client could be used which already operates through a network and is available cross-platform then the factoring of the application begins to look very attractive.

The ‘standard client’ being considered is any web browser that implements at least HTML level 2 and hence supports form-based interaction.

The minimal effort port is to replace the user interface functionality in the stand-alone application with an interface to an HTTP server that supports the common gateway interface (CGI), and to support HTML forms as input and HTML documents as output.

The maximal effort port is to make the server operate on multiple platforms and operate directly as an HTTP server.

2 Factoring an Existing Application into Client and Server

Figure 2 gives an overview of what is involved in factoring an application to operate in client-server mode through the web using existing web browsers as clients.

At the left the client component consists of a web browser providing a graphic-user interface through:

- *HTML documents* supporting typographic text
- *Images* supporting graphics, possibly embedded in the text
- *Hyperlinks* attached to text or images such as icons
- *Clickable maps* supporting returning to the server the location of a mouse click in an image
- *HTML forms* providing a programmable GUI returning data entered to the server.

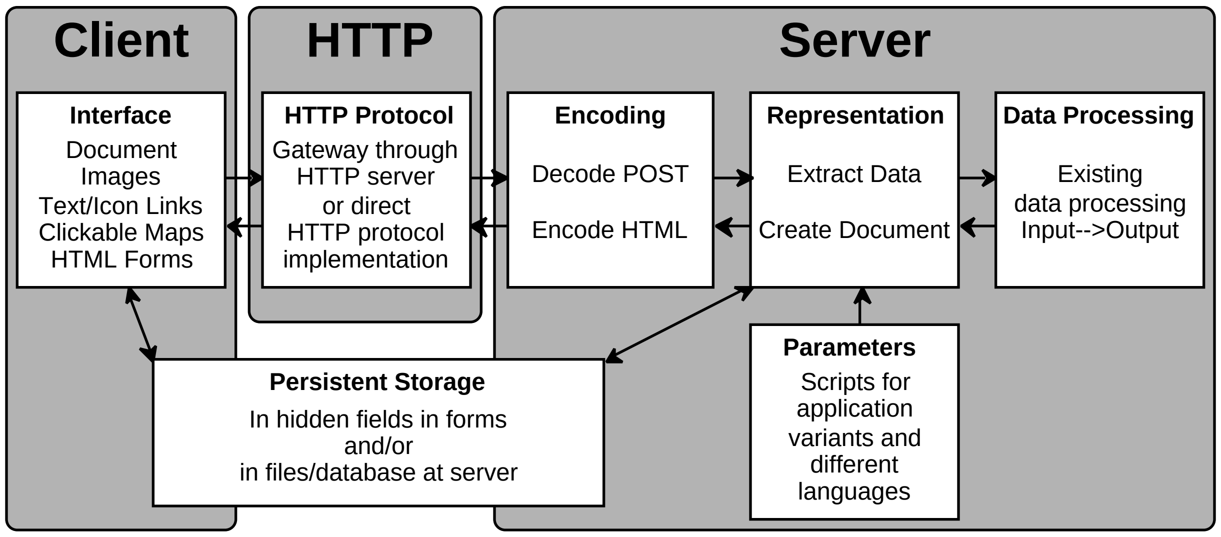


Figure 2 Factoring an application for client-server operation through the web

At the left center, the client communicates through the net with an HTTP server that is either an implementation of the HTTP protocol in the application server component, or a standard HTTP server providing a gateway to the application server component.

At the right the server component splits into 4 parts:

- Encoding the application output as HTML and decoding the incoming POST parameters
- Representation of the application output as a document and extracting incoming data
- Data Processing as the non-interface parts of the application
- Parameters as separate data structures or scripts providing application variants and dialog in different languages.

In addition, at the lower left the persistent storage for the application has to be factored to be represented either in hidden fields which can be stored as an HTML document at the client or in files or a database stored at the server, or some combination of both.

The following sections describe options and techniques for all these components and sub-components.

3 Restructuring Existing User Interfaces in HTML Forms

HTML level 2 forms capabilities provide a range of GUI widgets that can be embedded in web documents:

- Single and multi-line text entry boxes returning the text entered
- Radio and check boxes returning single and multiple selections, respectively
- Popup menus and selection lists returning single and multiple selections, respectively
- Clickable images returning the coordinates clicked
- Hidden data encoding the transaction state, not shown to the user but returned to the server.

These, together with normal HTML text, rules and images, allow a wide range of user interfaces to be encoded that, because they use the same widgets, can be made to resemble forms and dialog boxes in conventional applications.

3.1 Dialogs Programmed Using Hyperlinks Only

Dialogs that only select among alternatives can be programmed with text, images and hyperlinks. Figure 3 shows the initial dialog from a university information system that allows a user to select a service by clicking an icon or text hyperlink. It can be seen from the HTML generating it in Figure 4 that only simple hyperlinks are being used. Such an approach is often appropriate to the initial dialog in an application that accesses a number of distinct services.

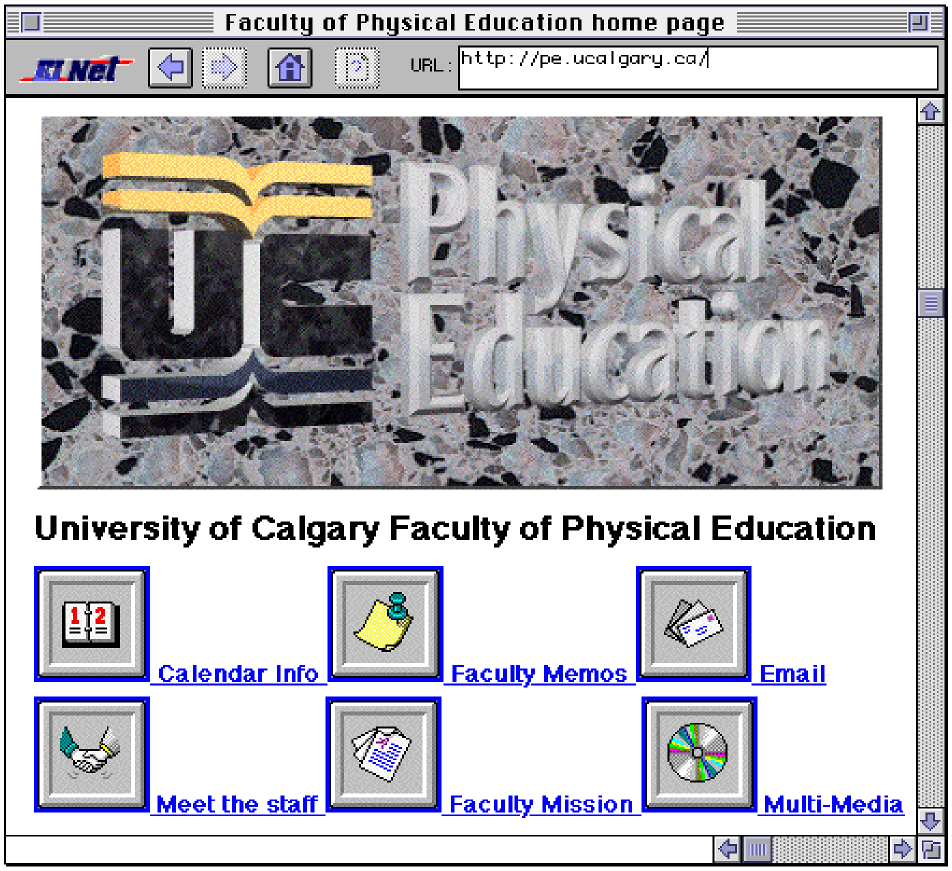

Faculty of Physical Education home page
URL: http://pe.ucalgary.ca/
Physical Education
University of Calgary Faculty of Physical Education
Calendar Info
Faculty Memos
Email
Meet the staff
Faculty Mission
Multi-Media

Figure 3 Interface to a university information system

```
<TITLE>Faculty of Physical Education home page</TITLE>
<IMG SRC="pelogo.gif">
<H1>University of Calgary Faculty of Physical Education</H1>
<B><A HREF="calendar.html">
<IMG SRC="cal.gif" ALT=" "> Calendar Info</A></B>
<B><A HREF="memos/readmem.html">
<IMG SRC="memo.gif"> Faculty Memos</A></B>
<B><A HREF = "execute/runmail.jcl">
<IMG SRC="mail.gif"> Email</A></B><BR>
<B><A HREF="meet.html">
<IMG SRC="meet.gif" ALT=" "> Meet the staff</A></B>
<B><A HREF="mission/mission.html">
<IMG SRC="mission.gif" ALT=" "> Faculty Mission</A></B>
<B><A HREF="media.html">
<IMG SRC="media.gif" ALT=" "> Multi-Media</A></B>
```

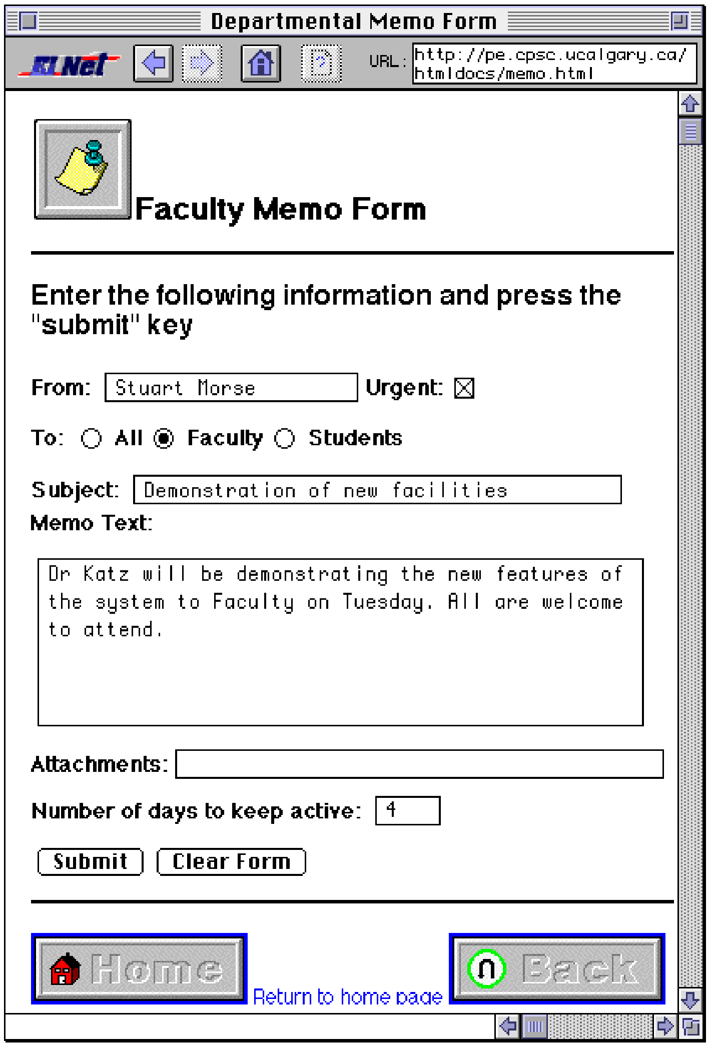
Figure 4 Document generating Figure 3

3.2 Using HTML Forms and Decoding POST Data from Forms

Clicking on the "Faculty Memos" icon in Figure 3 loads the document shown in Figure 5 which utilizes many features of HTML forms to provide transaction data entry facilities. The HTML generating this form is shown in Figure 6 and illustrates how text boxes, check and radio boxes and button are encoded.

Figure 7 shows the contents of the form as it is sent to the server in an encode form that replaces a space character with a "+" and uses "%" followed by a 2-digit character to encode non-alphanumeric data including "=" and "&" which are used to show the assignment of values in the form. Figure 8 shows this data decoded into a more readable format.

The decoding of the encoded data from a form is basically simple. The main complication is that forms may return multiple values for the same variable, for example multiple selections in a list. The decoder must be able to store a set of values for a variable, and application software must be able to cope with multiple valued selections if they are used. Multiple-valued variables are useful in many applications and it is worthwhile building them into the decoder from the outset.



Faculty Memo Form

Enter the following information and press the "submit" key

From: Stuart Morse Urgent: ☒

To: ○ All ● Faculty ○ Students

Subject: Demonstration of new facilities

Memo Text:

Dr Katz will be demonstrating the new features of the system to Faculty on Tuesday. All are welcome to attend.

Attachments:

Number of days to keep active: 4

Figure 5 Memo entry in a university information system

```
<HEAD><TITLE>Departmental Memo Form</TITLE></HEAD>
<BODY>
<H1><IMG SRC="memo.gif"> Faculty Memo Form</H1>
<HR>
<FORM METHOD="POST" ACTION="http://pe.ucalgary.ca/cgi-bin/post-memo">
<H2>Enter the following information and press the "submit" key</H2><BR>
<STRONG>From: </STRONG>
<INPUT SIZE=20 NAME="username">
<STRONG>Urgent:</STRONG>
<INPUT TYPE=CHECKBOX NAME="urgent" VALUE="yes"> <BR>
<STRONG>To: </STRONG>
<INPUT TYPE=RADIO NAME="to" VALUE="All" CHECKED>
<STRONG>All</STRONG>
<INPUT TYPE=RADIO NAME="to" VALUE="Faculty">
<STRONG>Faculty</STRONG>
<INPUT TYPE=RADIO NAME="to" VALUE="Students">
<STRONG>Students</STRONG>
<BR> <STRONG>Subject: </STRONG>
<INPUT SIZE=40 NAME="subject"> <BR>
<STRONG>Memo Text: </STRONG><BR>
<TEXTAREA NAME="Content" cols=50 rows=6></TEXTAREA> <BR>
<STRONG>Attachments:</STRONG>
<INPUT SIZE=40 NAME="attachments"><BR>
<STRONG>Number of days to keep active: </STRONG>
<INPUT SIZE=4 NAME="duration"> <BR>
<INPUT TYPE=SUBMIT VALUE="Submit">
<INPUT TYPE=RESET VALUE="Clear Form">
</FORM>
<HR>
<A HREF="../home.html"><iMG SRC="home.gif"> Return to home page</A>
<A HREF="memos/readmem.html"><iMG SRC="back.gif"> Go back to memo list</A>
</BODY>
```

Figure 6 Form generating user interface in Figure 5

```
username=Stuart+Morse&urgent=yes&to=Faculty&subject=Demonstration+of+new+facilities&
Content=Dr+Katz+will+be+domstrating+the+new+features+of%0D%0Athe+system+to+Faculty+
on+Tuesday.+All+are+welcome%0D%0Ato+attend.&attachments=&duration=4
```

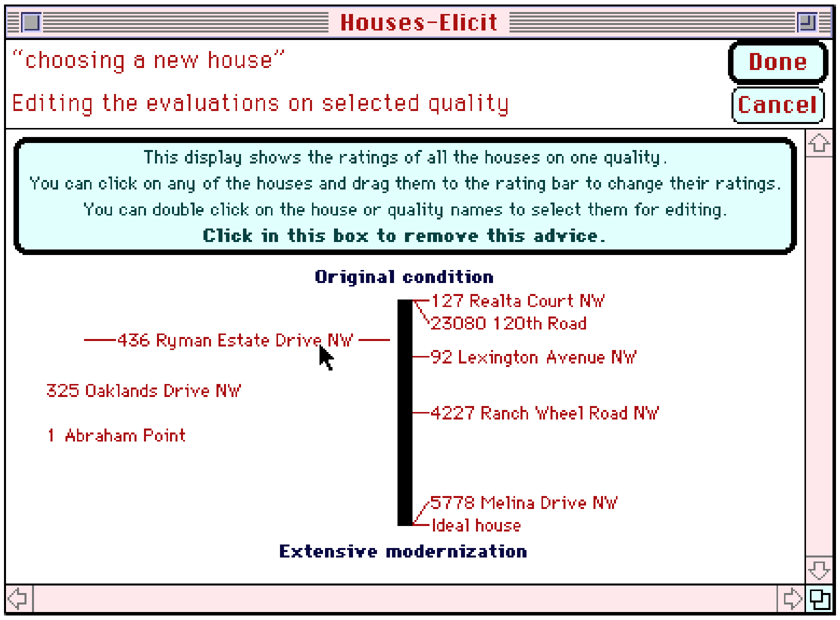
Figure 7 POST data returned by form of Figure 5

```
username = Stuart Morse
urgent = yes
to = Faculty
subject = Demonstration of new facilities
Content = Dr Katz will be domstrating the new features of
          the system to Faculty on Tuesday. All are welcome
          to attend.
attachments =
duration = 4
```

Figure 8 Decoded POST data returned by form of Figure 5

3.3 Redesigning Interfaces to Use HTML Form Widgets

Since the widgets currently available in HTML are limited and there is no facility for defining others, applications which make heavy use of special-purpose widgets or interactive graphics may be impossible to port. However, redesign of the interface around standard widgets is sometimes possible. Figure 9 shows a screen from our RepGrid/KSS0 (Gaines and Shaw, 1993) application which allows users to rate entities on a rating scale by dragging the entity name to a rating bar. This interface cannot be emulated directly in HTML. However, our WebGrid (Shaw and Gaines, 1995) application achieves the same functionality by using a popup menu in HTML as shown in Figure 10.



“choosing a new house”

Editing the evaluations on selected quality

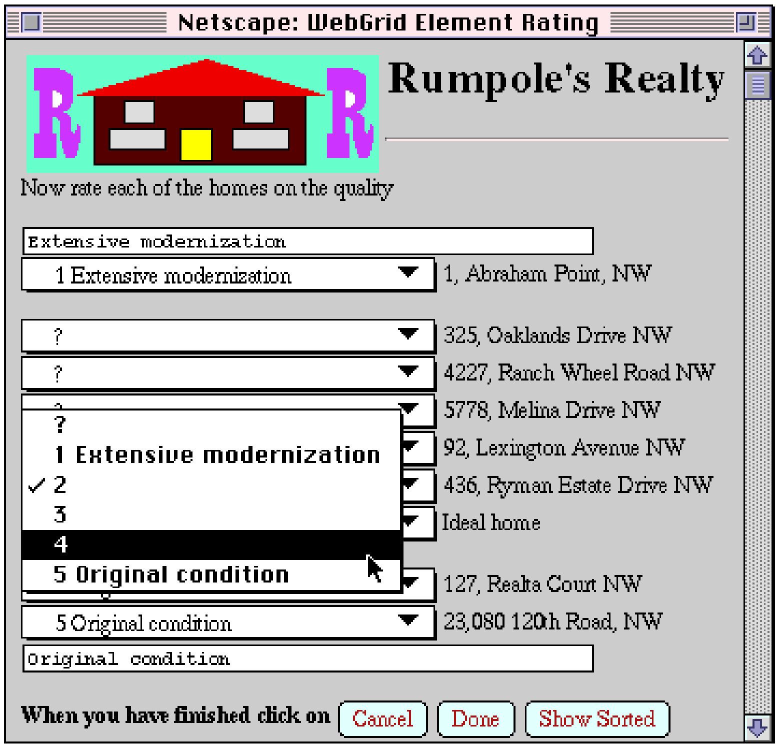
This display shows the ratings of all the houses on one quality.
You can click on any of the houses and drag them to the rating bar to change their ratings.
You can double click on the house or quality names to select them for editing.
Click in this box to remove this advice.

Original condition

127 Realta Court NW
23080 120th Road
436 Ryman Estate Drive NW
92 Lexington Avenue NW
325 Oaklands Drive NW
1 Abraham Point
4227 Ranch Wheel Road NW
5778 Melina Drive NW
Ideal house

Extensive modernization

Figure 9 RepGrid click and drag rating scale interface



Now rate each of the homes on the quality

Extensive modernization

1 Extensive modernization	1, Abraham Point, NW
?	325, Oaklands Drive NW
?	4227, Ranch Wheel Road NW
?	5778, Melina Drive NW
	92, Lexington Avenue NW
	436, Ryman Estate Drive NW
	Ideal home
	127, Realta Court NW
5 Original condition	23,080 120th Road, NW

Original condition

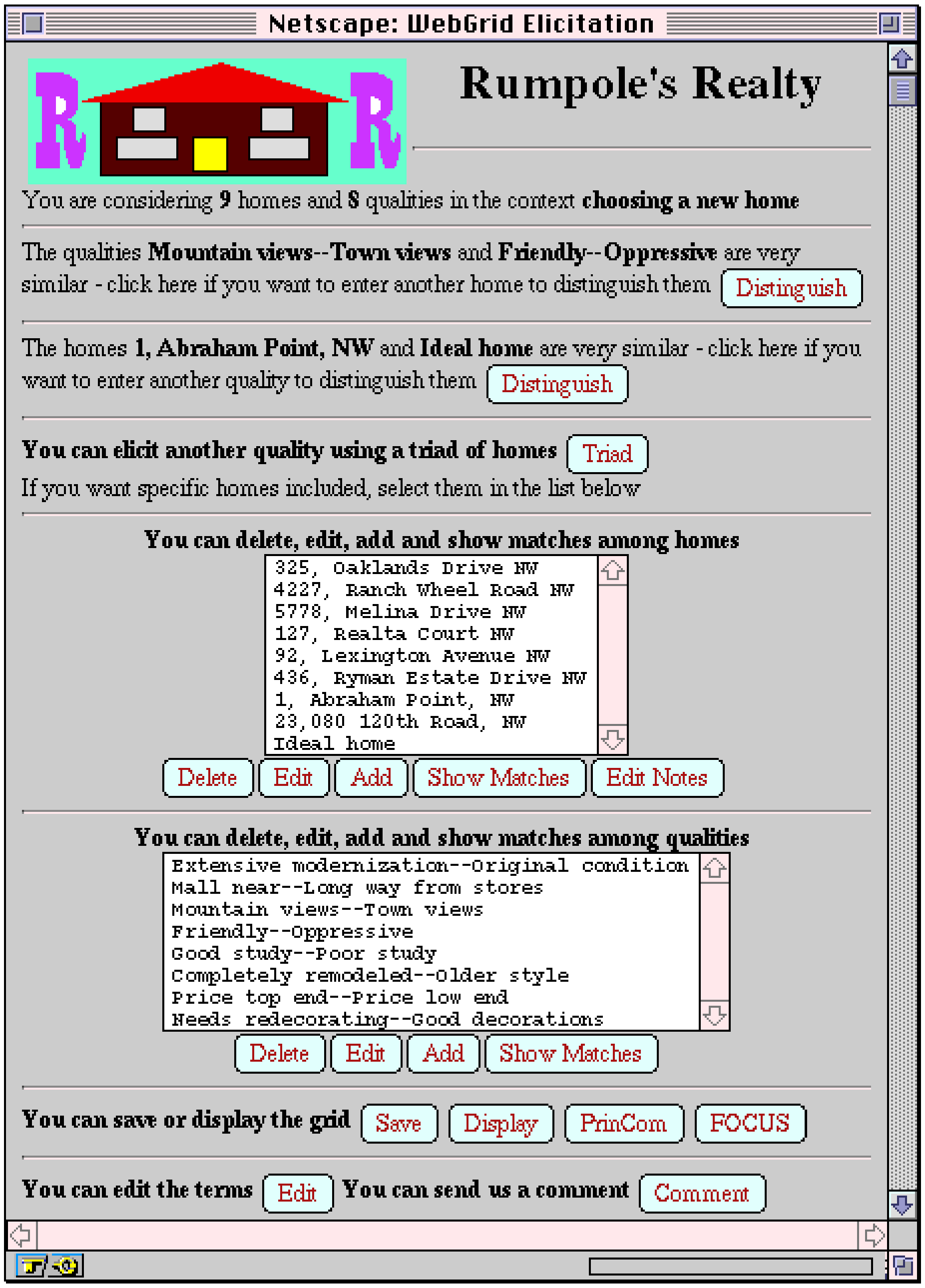
When you have finished click on

Figure 10 WebGrid click and drag rating scale interface

3.4 Reconceptualizing User Interfaces as Documents

Even though the user interfaces shown in Figures 3, 5 and 10 all appear much like GUIs in stand-alone applications, it is important in designing them to conceptualize them as *documents*. In particular, they will wrap if the user resizes the screen. It is difficult to design a form interface that looks good under all conditions of wrap. A good strategy is to design one for the default window width that will print on 8 inch wide paper, not be too concerned about narrower windows, but always use <BR> judiciously so that the layout is OK for wide windows.

Figure 11 is the main status screen from WebGrid, and illustrates some of the issues involved in conceptualizing a GUI as a document.



Rumpole's Realty

You are considering **9** homes and **8** qualities in the context **choosing a new home**

The qualities **Mountain views--Town views** and **Friendly--Oppressive** are very similar - click here if you want to enter another home to distinguish them [Distinguish]

The homes **1, Abraham Point, NW** and **Ideal home** are very similar - click here if you want to enter another quality to distinguish them [Distinguish]

You can elicit another quality using a triad of homes [Triad]

If you want specific homes included, select them in the list below

You can delete, edit, add and show matches among homes

```
325, Oaklands Drive NW
4227, Ranch Wheel Road NW
5778, Melina Drive NW
127, Realta Court NW
92, Lexington Avenue NW
436, Ryman Estate Drive NW
1, Abraham Point, NW
23,080 120th Road, NW
Ideal home
```

[Delete] [Edit] [Add] [Show Matches] [Edit Notes]

You can delete, edit, add and show matches among qualities

```
Extensive modernization--Original condition
Mall near--Long way from stores
Mountain views--Town views
Friendly--Oppressive
Good study--Poor study
Completely remodeled--Older style
Price top end--Price low end
Needs redecorating--Good decorations
```

[Delete] [Edit] [Add] [Show Matches]

You can save or display the grid [Save] [Display] [PrinCom] [FOCUS]

You can edit the terms [Edit] **You can send us a comment** [Comment]

Figure 11 The GUI as a document—main status screen in WebGrid

The main design principles involved are:

- A major chunk of functionality has been made available in a non-modal way so that the user does not have to fetch additional HTML documents from the server just to select functionality
- The main features of the current state of the application are displayed so that the user is able to make decisions about the next transaction without having to fetch additional documents
- The various functions available are split into small modules separated by horizontal rules so that the user can keep a complete sub-context in view even if they have to scroll through the document
- The various functions are being generated from the current state of the application in such a way as to be as helpful as possible in making clear the options available
- Some of the functions are optional and will not appear unless appropriate—there is no fixed layout.

4 Managing a Stateless Protocol—Partitioning Persistent Storage

The HTTP protocol is stateless so that no data is stored at the server unless the application itself does so. Most applications require persistent storage, and there are two techniques to achieve this:

- Store the persistent data in hidden fields in forms so that it persists at the client
- Store the persistent data in files/databases at the server and store one or more links (magic cookies) to this in a hidden field in forms.

4.1 Storing Data in Hidden Fields

The advantages of storing all the data in the forms are:

- No data is stored at the server and no management of persistent data is necessary
- Data filing can be supported at the client by saving the source of the HTML form
- The user can use the “Back” command in the browser freely as a multiple-level “Undo” without any special support at the server.

The primary disadvantage is that as the amount of data to be stored becomes large the transmission delays will increase until they are unacceptably high. This depends on the speed of transmission. A 14.4 KBaud modem with compression will transmit about 3KByte a second of hidden data so a reasonable limit is about 6KByte in a form.

Data stored in a hidden field is limited by the HTML DTD to be not more than 1024 characters long and should be chunked into several fields to be less than this (although most browsers do not enforce this limitation). The data structure usually leads to sensible chunking. However, large numbers of very small chunks should be avoided otherwise the overhead of the syntax of a hidden field can become excessive.

The repertory grid data for WebGrid is entirely encoded in hidden fields. Figure 12 shows the HTML generating Figure 11 so that the hidden data can be seen. It can be seen that the character “|” has been used as a separator to allow multiple data items to be stored in a single field. The server removes this character in data entered by the user.

Note that the server has returned a full pathname to itself in the form ACTION. This is important in enabling the user to store the data by storing the HTML source in a file at the client. When the form is loaded locally and user action is recommenced it will reconnect to the server.

```
<HEAD><TITLE>WebGrid Elicitation</TITLE></HEAD><BODY>
<IMG SRC="/RealEstate/RR.gif" ALIGN="left">
<H1 ALIGN="center">Rumpole's Realty</H1>
<HR><BR CLEAR="left"><FORM ACTION="http://tiger.cpsc.ucalgary.ca/KSS.acgi" METHOD=POST>
<INPUT TYPE="hidden" NAME="WebGrid" VALUE="10|01397738446|10/22/95|5:47:50 PM">
<INPUT TYPE="hidden" NAME="Header" VALUE="{IMG SRC=\/RealEstate/RR.gif\ ALIGN=\left\}
{H1 ALIGN=\center\}Rumpole's Realty{/H1}
{HR}{BR CLEAR=\left\}">
<INPUT TYPE="hidden" NAME="Head" VALUE="1|0|9|8|9|R1 5|">
<INPUT TYPE="hidden" NAME="Labels" VALUE="John & Mary|Homes|choosing a new home|John &
    Mary||quality|qualities|home|homes">
<INPUT TYPE="hidden" NAME="C0" VALUE="R|1 5||Extensive modernization||Original condition">
<INPUT TYPE="hidden" NAME="C1" VALUE="R|1 5||Mall near||Long way from stores">
.............................................
<INPUT TYPE="hidden" NAME="E0" VALUE="|0|4|1|0|0|0|0|4|325, Oaklands Drive NW|{P
    ALIGN=\CENTER\}{IMG SRC=\/RealEstate/H107.gif\}
{IMG SRC=\/RealEstate/R324.gif\} {BR}
Drive needs resurfacing. Mary--kitchen has a lovely view.
Good master bedroom? {BR}">
.............................................
<INPUT TYPE="hidden" NAME="CPair" VALUE="7|-32766">
<HR>The qualities <B>Needs redecorating--Good decorations</B> and <B>Town views--Mountain views</B>
    are very similar -click here if you want to enter another home to distinguish them
<INPUT TYPE="Submit" NAME="CMatch" VALUE="Distinguish">
<INPUT TYPE="hidden" NAME="EPair" VALUE="4|8">
<HR>The homes <B>92, Lexington Avenue NW</B> and <B>Ideal home</B> are very similar -
click here if you want to enter another quality to distinguish them
<INPUT TYPE="Submit" NAME="EMatch" VALUE="Distinguish">
<HR><B>You can elicit another quality using a triad of homes</B>
<INPUT TYPE="Submit" NAME="Triad" VALUE="Triad"><BR>
If you want specific homes included, select them in the list below
<CENTER><HR><B>You can
delete, edit, add
and show matches among homes</B>
<BR><SELECT NAME="Elements" MULTIPLE SIZE="9">
<OPTION VALUE="0">325, Oaklands Drive NW
<OPTION VALUE="1">4227, Ranch Wheel Road NW
.............................................
</SELECT><BR>
<INPUT TYPE="Submit" NAME="EDelete" VALUE="Delete">
<INPUT TYPE="Submit" NAME="EEdit" VALUE="Edit">
.............................................
<CENTER><HR><B>You can
delete, edit, add
and show matches among qualities</B>
<BR><SELECT NAME="Constructs" MULTIPLE SIZE="8">
<OPTION VALUE="0">Extensive modernization--Original condition
<OPTION VALUE="1">Mall near--Long way from stores
.............................................
</SELECT><BR>
<INPUT TYPE="Submit" NAME="CDelete" VALUE="Delete">
<INPUT TYPE="Submit" NAME="CEdit" VALUE="Edit">
<INPUT TYPE="Submit" NAME="CAdd" VALUE="Add">
<INPUT TYPE="Submit" NAME="CShow" VALUE="Show Matches"></CENTER>
<HR><B>You can save or display the grid</B>
<INPUT TYPE="Submit" NAME="Save" VALUE="Save">
.............................................
<HR><B>You can edit the terms</B>
<INPUT TYPE="Submit" NAME="GEdit" VALUE="Edit">
<B>You can send us a comment</B>
<INPUT TYPE="Submit" NAME="Comment" VALUE="Comment">
</FORM></BODY>
```

Figure 12 Form generating user interface in Figure 11

4.2 Storing Data at the Server

If the application involves large datasets or updating databases to be maintained at the server, it is not possible to store all the data in HTML forms. One can maintain state information within a series of connected transactions by storing unique transaction identifiers in the forms. The value 01397738446 in line 5 of Figure 12 is such a UID, not used by WebGrid but generated automatically by the generic server class in our C++ class library from which the WebGrid server is subclassed.

Persistent storage at the server creates many problems:

- Temporary data that persists only for a series of transactions must be purged taking into account that transactions may be discontinued in mid-stream
- Long-term data filing at the server must be controlled, for example through an authorization system
- The effect of the “Back” command in the browser must be carefully managed since the user may step back in a series of transactions and attempt to take actions based on state information that is out of date, and an error message should be generated to indicate this
- If multiple users can access the data stored at the server then appropriate transaction interlocks must be provided.

These problems are present in conventional client-server systems and can be solved within a transaction processing framework. However, if one is just porting an interactive application to the web it is worth avoiding them as much as possible. One should not assume that state information has to be stored at the server. As much of the information as possible should be stored in hidden data in forms and server storage used only when it is essential to the application.

In many applications it is appropriate to combine the two techniques, storing all transaction data in a form until the transaction is complete and then allowing the user to store it at the server. For example, in WebGrid grids need to be uploaded to the server to make them available for comparison with others but this is only done when the series of transactions involved in eliciting the grid are complete.

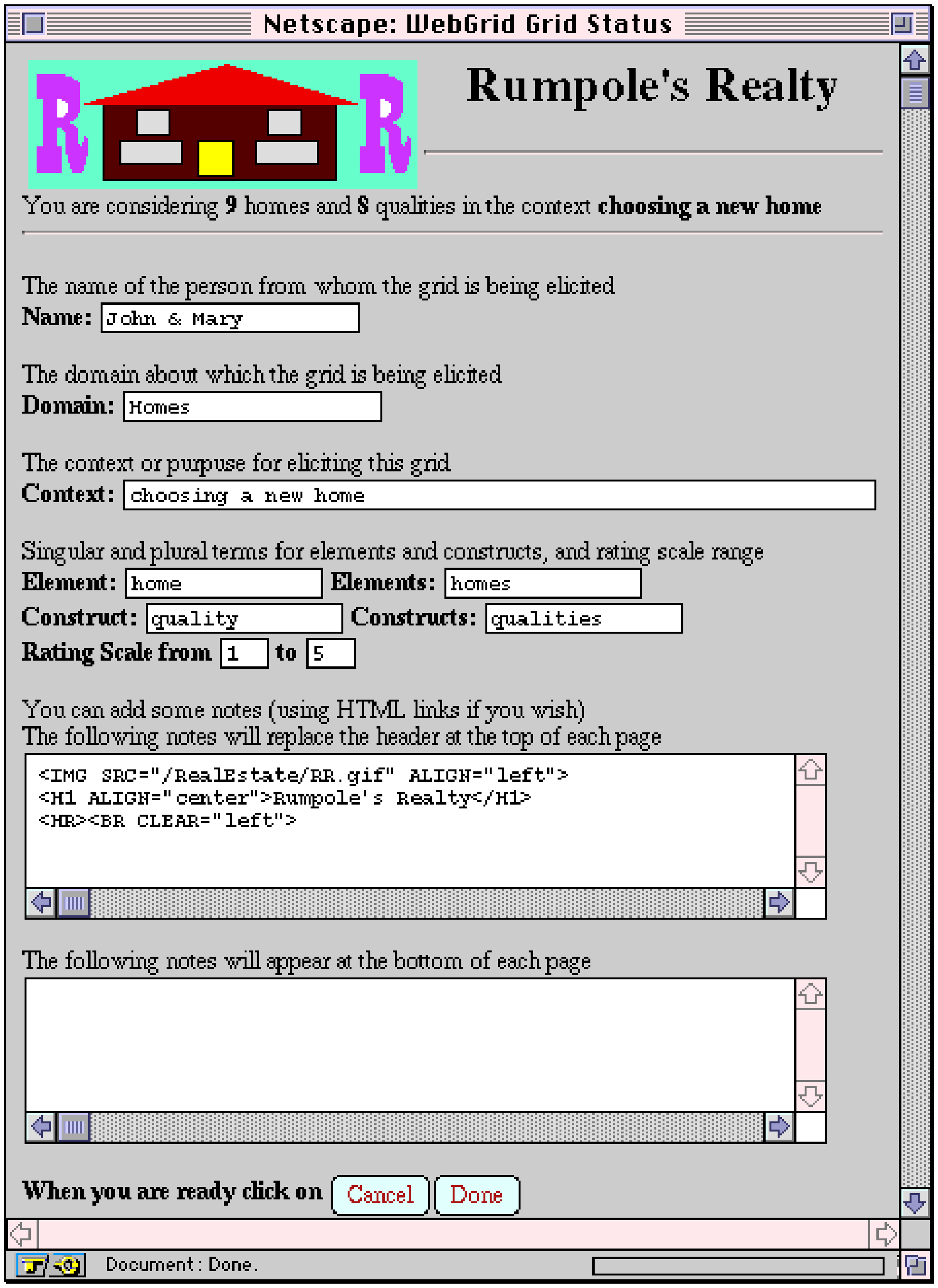
4.3 Encoding Dynamic HTML Data from User for Display, Editing and Hidden Fields

To take full advantage of an application running on the web it is useful to be able to support the user entering HTML data which will be inserted in the output in appropriate places. For example, the “Rumpole’s Realty” header in Figures 10 and 11 is generated through user customization of the WebGrid header. The text box second from the bottom in Figure 13 shows the data to do this being made available for entry and editing.

User entered dynamic HTML data has to be translated between three different forms:

- Raw HTML which is received as POST data from the text box and output in other forms as part of the form
- Encoded HTML which is output to the text box for editing purposes and is to be interpreted as character data and not as tags, so that, for example, the “<“ character needs to be encoded as the string “&lt;”
- Encoded HTML which is stored in hidden fields and must neither be interpreted as character data nor as tags, so that some alternative character must be substituted for “<“, “>“ and “&”—WebGrid uses “{“, “}” and “\” respectively, and does not allow these in data entry.

It is simple to write routines that translate between these formats and offer the user the opportunity to enter HTML data, such as personal notes and links to images, that is used as part of the application data.



Rumpole's Realty

You are considering **9** homes and **8** qualities in the context **choosing a new home**

The name of the person from whom the grid is being elicited
Name: John & Mary

The domain about which the grid is being elicited
Domain: Homes

The context or purpose for eliciting this grid
Context: choosing a new home

Singular and plural terms for elements and constructs, and rating scale range
Element: home **Elements:** homes
Construct: quality **Constructs:** qualities
Rating Scale from 1 **to** 5

You can add some notes (using HTML links if you wish)
The following notes will replace the header at the top of each page

```
<IMG SRC="/RealEstate/RR.gif" ALIGN="left">
<H1 ALIGN="center">Rumpole's Realty</H1>
<HR><BR CLEAR="left">
```

The following notes will appear at the bottom of each page

When you are ready click on Cancel Done

Figure 13 HTML data entered and edited by users to customize output

5 Graphic Output

If the original applications displays some of its output in graphic form, it is simple to convert this to a format that can be read by web browers. It is often appropriate to use the graphic as a clickable map.

5.1 Graphic Formats

The two formats that can be read inline without client helpers by browsers such as Netscape are GIF and JPEG. The Compuserve GIF format is information lossless and will reproduce pixel for pixel the exact form of the original graphic, and is the preferred format for most graphic output. It is restricted to a 256-color palette, but this is the most common color capability at browsers. One has to be careful not to rely on use of higher color resolution unless one is running a specialist service to known equipment.

The GIF format uses Lempel-Ziv run-length encoding of horizontal scan lines to compress the image structure. This scheme is very effective at compressing pictures comprised of long horizontal segments of identically colored pixels, and very poor at encoding images involving pixel-by-pixel changes along such a scan line. This means that it is not the absolute size of a picture that determines the data that must be transmitted, but rather the horizontal complexity of the image. Vertically, a graded tint that changes pixel by pixel causes no problems. Thus one may include large colored images provided the horizontal color changes are few.

JPEG is useful if the source is photographic, generally giving better compression on true images.

5.2 Converting Graphic Output to GIF Format

Algorithms for converting a color bit map to GIF and JPEG formats are simple and well-documented in many books. FAQs and source code are available on the net. For use in an online service it is very important to optimize the algorithm for speed. Simply written algorithms can take tens of seconds to do the conversion, whereas current RISC processors can convert large images, say 500 by 300, in less than 1 second.

GIF conversion is a pixel by pixel operation and its speed is dominated by one small inner loop that takes 99% of the computation time. Careful optimization of this loop can result in substantial speedups, and if image output is a substantial part of the application then converting the loop to native instructions is worth considering.

Luse (Luse, 1993) provides useful starting code for GIF conversion. It is written in C++ but there is no advantage to an OO approach and it is best stripped back to C. The code supplied is not portable because it writes out data structures as if they were character strings not taking account of whether a machine is little- or big-endian in its byte sequences. However, this is easily changed, and the hash schema for the Lempel-Ziv conversion is very efficient. The pixel by pixel conversion loop should be converted from array accesses to pointer accesses for better speed with most compilers and optimized for register usage.

Figure 14 shows some colored graphic output generated from a WebGrid data analysis sent to Netscape and saved as a GIF image. The 467 by 304 pixel image occupies 5 KByte. This image is not stored on disk at the server but delivered directly to Netscape using the HTTP code shown in Figure 15.

The HTTP protocol does not currently support multi-part MIME documents so that if one wishes to embed a generated graphic in a document then one must store either the graphic itself or a data structure sufficient to generate it on disk at the server. An IMG URL embedded in a document can then point to the graphic file or the data file with information on its generation. The only disadvantage of this is that provision must be made to purge these files when they are no longer needed. Since a user may leave a browser with transaction data in the middle of a transaction and come back to it days later and attempt to back to a document containing an image, there is no simple answer to how often purging should take place. By putting appropriate status information, in the URL it is simple to generate an error message indicating that the server cache has expired.

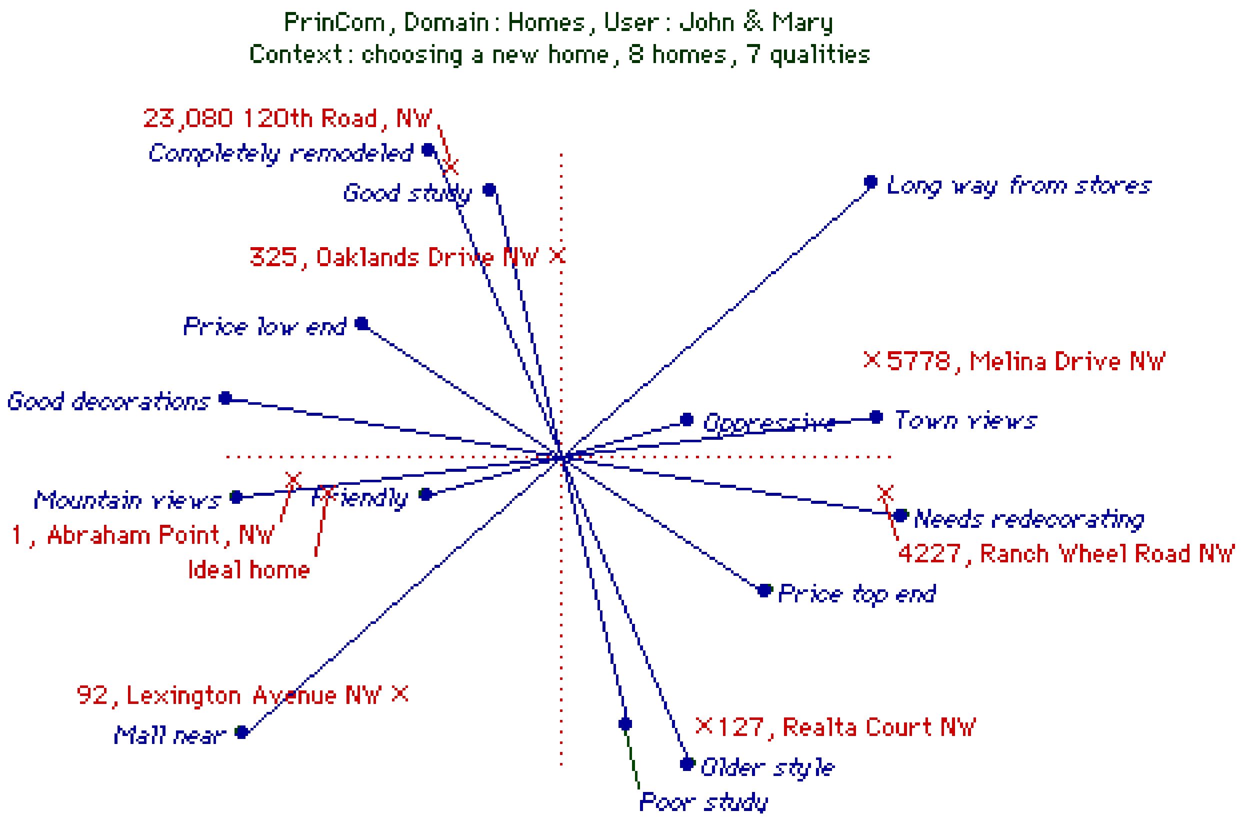


Figure 14 467 x 304 graphic occupying 5KByte

```
HTTP/1.0 200 OK
Server: WebGrid
MIME-Version: 1.0
Content-type: image/gif
Content-length: 4998

GIF87ah..................................
.................................................
```

Figure 15 HTTP message to browser generating Figure 14

5.3 Dynamic Clickable Maps as User Interface Elements

A very powerful technique is to use graphics generated dynamically as interactive user interface elements by making them clickable maps. Since the server generates the graphic it is simple to keep the data structure at the server and use it to interpret a click in the graphic sent back as POST data. We use this technique in our implementation of concept mapping tools on the web (Gaines and Shaw, 1995).

Figure 16 shows a concept map generated from a data structure stored at the server, and Figure 17 shows the HTML generating it. When the server receives the request for an image whose source URL is "http://tiger.cpsc.ucalgary.ca./WebMap/Web66.k" it loads the Web66 file, converts it to a GIF and send it to the client. When it receives the click coordinates of a click in a map sent to URL "http://tiger.cpsc.ucalgary.ca./WebMap/Web66.k" it loads the Web66 file again and determines in which node the click occurs. It then looks up the command attached to the node and executes it at the server, in this case to redirect the browser to some other URL.

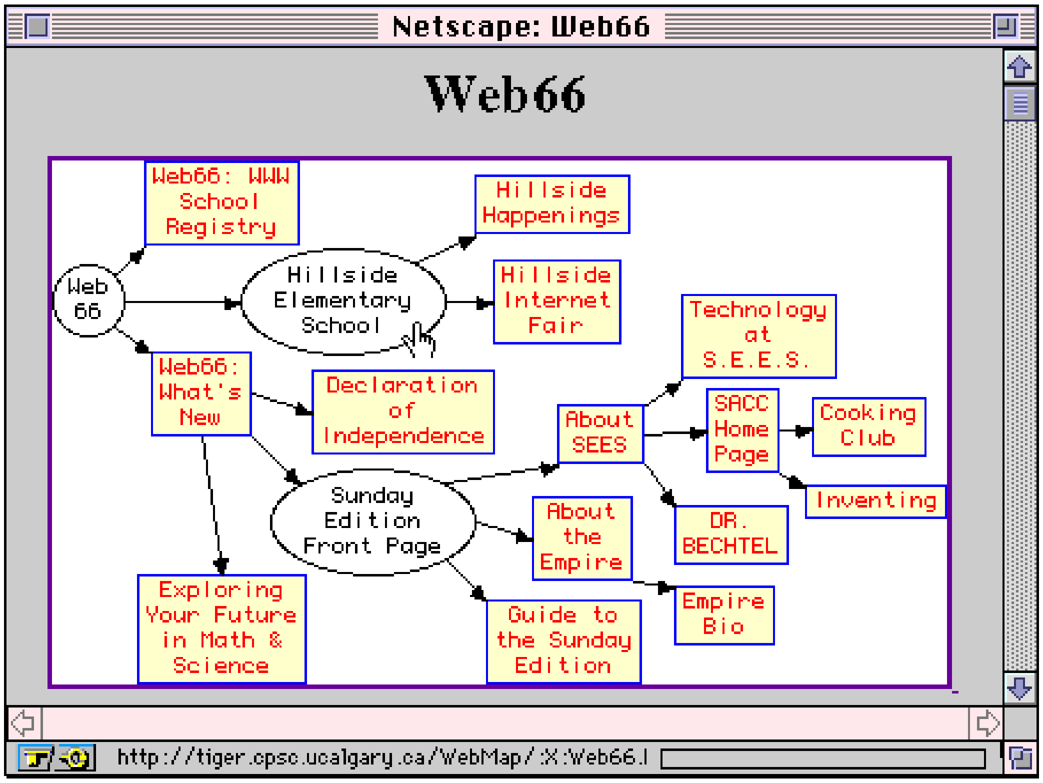

Netscape: Web66
Web66
Web66: WWW School Registry
Hillside Happenings
Web 66
Hillside Elementary School
Hillside Internet Fair
Technology at S.E.E.S.
Web66: What's New
Declaration of Independence
About SEES
SACC Home Page
Cooking Club
Sunday Edition Front Page
About the Empire
DR. BECHTEL
Inventing
Exploring Your Future in Math & Science
Guide to the Sunday Edition
Empire Bio
http://tiger.cpsc.ucalgary.ca/WebMap/:X:Web66.l

Figure 16 Dynamically generated clickable map

```
<HEAD><TITLE>NetscapeAgent</TITLE></HEAD>
<BODY>
<H1 ALIGN="CENTER">NetscapeAgent</H1>
<P ALIGN="CENTER">
<A HREF="http://tiger.cpsc.ucalgary.ca./WebMap/Web66.k">
<IMG SRC="http://tiger.cpsc.ucalgary.ca./WebMap/Web66.k" ISMAP>
</A>
</BODY>
```

Figure 17 HTML generating clickable map of Figure 16

6 Designing for Multiple Languages and for Customization

It has become standard practice in interactive computer applications to embed all user dialog in some form of “resource file” that can be edited quite separately from the application code. This enables applications to be customized for different purposes and localized for use in different languages by editing the resource file without having to recompile the code.

It is worthwhile taking a similar approach to client-server systems on the web, and putting all user dialog into separate script files that can be edited in a text editor. This allows the server to operate in multiple languages based on a simple parameter embedded in the form with values, “English”, “French”, “German” and so on. The script files can usually take the form of HTML with embedded macros that are expanded from parameters set up by the code.

Figure 18 shows the WebGrid script generating the HTML which generates the status screen of Figure 13. The backslash character “\” is used to demarcate the script sections and to introduce macro codes. A line containing only a backslash is a section terminator. A line containing a double backslash followed by a letter starts a section named by that letter. In the code the start of a new section causes a case statement jump to a routine associated with that letter.

```
WebGrid Grid Status\
<IMG SRC="http://\S/WebGrid/WebGrid.gif" ALIGN="left" ALT="WebGrid Icon">
<H1 ALIGN="center">WebGrid <I>Grid Status</I></H1><HR>
\
\\N
You are considering <B>\0</B> \1 and <B>\2</B> \3 in the context <B>\D</B><HR>
\
\\U
<BR>
\#0The name of the person from whom the grid is being elicited\
\#Enter your name\
<BR><B>Name: </B><INPUT NAME="User" VALUE="\u"><BR><BR>
\
\\F
The domain about which the grid is being elicited<BR>
<B>Domain: </B><INPUT NAME="Domain" VALUE="\d"><BR><BR>
The context or purpuse for eliciting this grid<BR>
<B>Context: </B><INPUT NAME="Context" VALUE="\D" SIZE=60><BR><BR>
Singular and plural terms for elements and constructs, and rating scale range<BR>
<B>Element: <INPUT NAME="ETerm" VALUE="\e" SIZE=15>
Elements: </B><INPUT NAME="ETerms" VALUE="\E" SIZE=15><BR>
<B>Construct: <INPUT NAME="CTerm" VALUE="\c" SIZE=15>
Constructs: </B><INPUT NAME="CTerms" VALUE="\C" SIZE=15><BR>
<B>Rating Scale from <INPUT NAME="Low" VALUE="\r" SIZE=3>
to </B><INPUT NAME="High" VALUE="\R" SIZE=3>
\#0<P>You can add some notes (using HTML links if you wish)<BR>
The following notes will replace the header at the top of each page<BR>
<TEXTAREA NAME="Heading" ROWS=5 COLS=60>\h</TEXTAREA>
<P>The following notes will appear at the bottom of each page<BR>
<TEXTAREA NAME="Trailing" ROWS=5 COLS=60>\H</TEXTAREA>\
\#\
\
\\X
<HR><BR>
\#0You will be asked to enter your ratings for each \e on each \c in this grid\
\#You will be asked to develop your own \C based on the \E in this grid\
\#You need to enter some \E which you will be asked to rate on each \c in this grid<BR>
<TEXTAREA NAME="Elements" ROWS=10 COLS=60></TEXTAREA>\
\
\\E
<BR><BR><B>When you are ready click on</B>
\#0<INPUT TYPE="Submit" NAME="Cancel" VALUE="Cancel">\
\#\
<INPUT TYPE="Submit" NAME="OK" VALUE="Done">
\
\\
```

Figure 18 WebGrid script for the status screen of Figure 13

In the script text a backslash followed by a number indicates a parameter to be substituted by some data already set up in the code. A backslash followed by a "#" and a number introduces a list of possible strings to be selected on the basis of a numeric parameter set up by the code.

It is simple to develop a macro substitution language that can be used to parameterize the output from a range of applications. The effort is amply repaid by the capability to change the user interface without recompilation, and to customize the application.

7 Impact of Enhancements of Web Functionality

This tutorial has been based on what may be achieved with browsers that support HTML level 2. As support for HTML tables becomes common it is giving greater control over the layout of GUIs based on HTML forms. Netscape's "frames" if widely adopted will give even greater control.

The greatest impact on user interface functionality will come if the loading of code to a browser to be executed locally becomes widely supported. The existing demonstrations of Sun's (Sun, 1995) "HotJava" capabilities show that highly interactive graphic user interfaces can be supported through downloading the appropriate code.

Acknowledgments

The system shown in Figures 3 and 5 was programmed by Stuart Morse according to requirements specified by Larry Katz and Bill Richardson as part of an MSc project supervised by Mildred Shaw. I am grateful to Mildred Shaw for detailed comments on the first draft of the tutorial notes.

References

Gaines, B.R. and Shaw, M.L.G. (1993). Eliciting knowledge and transferring it effectively to a knowledge-based systems. **IEEE Transactions on Knowledge and Data Engineering 5**(1) 4-14.

Gaines, B.R. and Shaw, M.L.G. (1995). WebMap: concept mapping on the web. **World Wide Web Journal 1**(1) 171-183.

Luse, M. (1993). **Bitmapped Graphics Programming in C++**. Reading, Massachusetts, Addison-Wesley.

Shaw, M.L.G. and Gaines, B.R. (1995). Comparing constructions through the web. Schnase, J.L. and Cunnius, E.L., Ed. **Proceedings of CSCL95: Computer Support for Collaborative Learning**. pp.300-307. Mahwah, New Jersey, Lawrence Erlbaum.

Sun (1995). HotJava Home Page. Sun Microsystems. http://java.sun.com.

URLs

WebGrid can be accessed at **http://tiger.cpsc.ucalgary.ca/WebGrid**

Related papers on WebGrid, WebMap and World Wide Web can be accessed through **http://ksi.cpsc.ucalgary.ca/KSI**