View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Loughborough University Institutional Repository

This item was submitted to Loughborough's Institutional Repository by the author and is made available under the following Creative Commons Licence conditions.

creative commons

COMMONS DEED

Attribution-NonCommercial-NoDerivs 2.5

You are free:

- to copy, distribute, display, and perform the work

Under the following conditions:

Attribution. You must attribute the work in the manner specified by the author or licensor.

Noncommercial. You may not use this work for commercial purposes.

No Derivative Works. You may not alter, transform, or build upon this work.

- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

This is a human-readable summary of the Legal Code (the full license).

Disclaimer

For the full text of this licence, please go to:
http://creativecommons.org/licenses/by-nc-nd/2.5/

A HIGHLY CONFIGURABLE QUERY-ORIENTED PORTAL FOR A CO-OPERATIVE ENVIRONMENT

Roger Stone

Department of Computer Science, Loughborough University, UK
r.g.stone@lboro.ac.uk

ABSTRACT

Web portals and "my" portals are now commonplace but they are constructed along familiar lines with hierarchical management structure. Typically one or more owners of data will allow a larger group of people to view data in rigid, pre-planned ways. Historically data on the web was presented on static pages. Nowadays the data is likely to be drawn from a database, but it still tends to be presented in a layout defined by the supplier. This suggests that it might be fruitful to consider what designs might be natural or possible in a cooperative, trusting, egalitarian environment. Although this ideal situation is rarely, if ever, truly found in real life it may be that some of the ideas will be appealing and find application even in an imperfect environment.

KEYWORDS

Portal, User-Interface, Web Query, Web Update, Database, Co-operation, Homepage Editor

1. INTRODUCTION

Web portals are now commonly available and most web users will have had the opportunity to use one either at the workplace via a corporate portal or at home via an ISP or the "my" offerings from search engines (myYahoo [1], myGoogle [2]). The facilities offered by customisable portals have been studied in various ways. For example Manber *et al.* [3] discuss experiences with an individual "my" portal, Bellas [4] reports a flexible framework for constructing portals, Wege [5] discusses the technology required and Perugini & Ramakrishnan [6] survey the way in which the personalizing of interactions is handled across a wide variety of systems. Priebe [7] discusses the typical lack of integration among links (portlets) available from the homepage.

All customisable portals offer the user some degree of personalisation of the interface. There are choices to be made about what appears on the user's home page and where it appears. However the models used by these portals are based on traditional, very hierarchical views. Typically one individual (the webmaster) controls the provision of new data or, if ordinary users are allowed to upload data, then they have their own segregated areas to which they have authorship. It is the intent of this paper to explore the possibilities of what a portal might become in a more cooperative environment where a significant proportion of the data available at the portal (if not all) is regarded as shared and editable by all. In a truly cooperative environment all users would be treated equally and would share the ownership of all data. Viewed against the background of equality of users it is not acceptable to have a webmaster prescribing who can have access to data, how the data may be viewed, who can edit existing data and who can add new data. This is consistent with Dignum [8] who argues for a shift in the focus of Knowledge Management to encourage personal participation and foster an atmosphere of trust.

From this point of view we can explore whether such a cooperative portal can actually be constructed and, if it is possible, whether it can serve its users better than the current hierarchical products. From a simplistic viewpoint a portal consists of a homepage and a large collection of links to other pages or services. The homepage will typically be partly customisable allowing the user to choose which links to have on view and possibly to control the position and style of their appearance on the homepage.

The presentation of material in the paper breaks down into two main parts. Firstly there is a discussion of the facilities which might be offered to the user to control the look of the portal, i.e. the facilities of the homepage editor. Secondly there is a discussion of what control the user has over the pages that might be available via links from the homepage. Instead of selecting from prescribed views of selected data provided by a webmaster, the users ought to be able to create dynamic pages themselves by selecting freely any combination of data and choosing the presentation and styling. In addition to freedom of viewing, the users should be trusted and be given the ability to edit any data on view.

The example scenario that has initially motivated this approach is the situation of staff in a University department who have access to an intranet facility [9]. The staff maintain teaching and administrative data about the students in their care. Whilst it is true that a staff member might possess certain information about a student that is confidential, there is much to be gained by sharing as much data as possible and fostering a community ownership and responsibility for the data. It may be the case that in practise a truly cooperative community with no secrets does not exist, but nevertheless it is reasonable to explore in that direction and see what opportunities and benefits might arise.

2. THE PORTAL HOMEPAGE AND ITS EDITOR

Typically the portal homepage carries the corporate title and logo and uses corporate colours. The user's name or designation will appear and there will be a section of the page which can be 'edited' by a specific homepage editor which will allow the user to adjust what they see on the homepage and may allow some adjustment of the layout. For example an editor may allow the user to choose a subset of a collection of links to various services to which they have access and the editor may provide these as a list which may be re-ordered at will.

Good examples of public portals can be found at myYahoo [1] and myGoogle [2]. They both have a great deal of customisable content but while myYahoo uses buttons to move sections of content up/down/left/right, myGoogle uses Javascript drag and drop interaction to rearrange content.

There is perhaps surprisingly little in the way of drag-and-drop interaction on web pages. The early adopters of dynamic HTML who tried to make drag-and-drop interactions on web-pages using Javascript found it very hard to write cross-browser code. At first mono-browser solutions were created, e.g. for Internet Explorer [10]. Later authors tried to write code that would work across Internet Explorer and Netscape Navigator browsers [11]. With the current range of W3C DOM compatible browsers the coding has become much simpler and is available as 'wizard' code in WYSIWYG web-page editors (*e.g.* Dreamweaver [12]). There are also Javascript code modules which allow the coding of drag-and-drop interactions in a straightforward way (*e.g.* Cozens [13]). It is the arrival of browsers that conform to the W3C standards that has made this programming much easier although it is still not much exploited.

To make a drag-and-drop link editor for a portal, essentially each link is contained in a tag (with a style attribute containing position information) and an onClick event handler. The handler tracks the mouse during the drag and continually repositions the link at the mouse

coordinates until the link is released. At the drop point the handler finishes by writing the new, final position of the link to the database. The notion of drag-and-drop editing is central to the variations of homepage editor presented below.

2.1. A Graphical Homepage

In order to give the user the most control over the look of their homepage, consider a blank canvas on which the user can position any link at any position, i.e. at any (x,y) coordinate on the screen. The homepage editor would allow the user to choose

- the text of the link
- an image (icon) for the link
- a style for the link
- a URL for the link
- a 2-dimensional graphics coordinate for the link

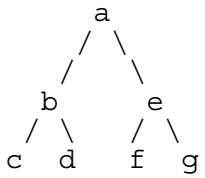
By adding the name (or some other identification) of the user, each link comprises 6 pieces of information and the links can conveniently be stored as records in a database table. The position of the link could be expressed as an (x,y) coordinate or (top,left) coordinate in pixels. Either the link text or the link image might be omitted but not both. The editor to support this might be form-based and allow the user to type in alterations and additions and make deletions. While this might be acceptable to change the text/icon/style/URL attributes of the link it would be very frustrating to use a text-based editor to alter the coordinate position. What is required is a graphical editor. There should be a button, link or menu item available via the normal view of the homepage which takes the user to a graphical editor in which the links are visible but now the click action does not take the user to the URL destination but instead allows the user to re-position the link by sliding it across the page to a new position. In this way the user has complete natural control over the positioning of the links.

Upon user-trialling of this graphical editing interface, two issues arose. Firstly the behaviour of the interface was so far removed from the normal expectation of users that many needed to be physically shown it in use before they fully grasped the idea. The second issue was that although users had the ability to place the links anywhere on screen, it was clear that most users were trying to group the links and align them in rows and columns. However the pixel coordinate system of modern screens is very fine and users had trouble lining up links horizontally and vertically into rows and columns. This was easily remedied by simulating a courser grid which was customisable by the user.

2.2. The list-based homepage

It has already been stated that it became clear on user trialling of the graphical editor that the grouping of links and presenting them in neat rows and columns was important to most if not all the users. Therefore it seemed worthwhile to look for another editing style that could be used to specify the link layout. The inspiration for the new idea was the various grouping presentation facilities normally to be found in web-pages. There is the paragraph, the div-ision, the table (giving a vertical presentation of rows), the table row (giving a horizontal presentation of entries), the bullet list (which can be nested) and the drop-down list. The intuition was that given any hierarchically grouped information then each level could be arbitrarily styled with any of the presentation tags yielding different looking layouts.

Thus, for example, a hierarchy with internal nodes representing (optional) labels and terminal nodes representing links that might be represented abstractly as



could be shown on a web page using (nested) (unordered) lists or as a table.

Heading a

- Heading b
  - Link c
  - Link d
- Heading e
  - Link f
  - Link g

Label a

Heading b	Link c	Link d
Heading e	Link f	Link g

The idea is that any node in the hierarchy can be allocated any of the following styling alternatives which provide a tag for the node and a companion tag for all its children.

Structure	***Node tag***	***Children tag***
table	<table>	<tr>
table row	<tr>	<td>
bullet list		
numbered list		
drop-down list	<select>	<option>
paragraph	<p>	(none)
division		(none)

Of course there are restrictions. For example, there is an obvious restriction regarding nesting or containment. Although a table cell can contain a table or any other object, a select option cannot contain any other list structure. Regarding consistency, if any node is styled as table, then the nodes which are its children have to be styled as table rows (<tr>) and the children's children as table cells (<td>).

In order to make this into an interface the concept was to allow the user to interact with the hierarchy in two ways

i) indicate styling on a per node basis
ii) move elements arbitrarily about the hierarchy

The styling would require that the user be able to click on a node and then choose one of the styling forms (perhaps by selecting from a drop down list). The moving of an element would require that the user be able to drag any subtree (a node with any descendants) and attach the node as a subnode of a node that is not being moved and to somehow signify its position among the existing children of the receiving node.

The visual editor for this idea was quite challenging. In order to give the user n+1 places to insert a new element into an n element display a thin graphic element was drawn before, between and after elements in a list to give a target for drag-and-drop. This meant that the editor was not WYSIWYG as these targets were removed when editing was complete. This

caused an associated shrinkage of the expanse of each list and therefore a significantly different overall display. Early user reaction was not favourable. The interface although appealing in theory was regarded as being very complicated to use. Consequently further development on this variant of editor was halted.

Drag-and-drop is still in its infancy on the web and there is a problem with users not expecting to find interfaces using it. Nevertheless it is now implemented satisfactorily and should be used routinely as it is in non-web applications.

3. USER-DEFINED DYNAMIC PAGES

Now that the homepage and its editor have been considered, we turn to the content of the links that might appear on the homepage. So far the homepage has been discussed as a selection from a given set of links offering views of data as prescribed by a webmaster. So for example within an organisation a link to a staff directory may be available which when followed brings up a page of staff telephone numbers and room numbers organised alphabetically by staff surname. It may well be that this page is a dynamic page that has been provided by the webmaster and derives its information from a database. But as far as the user is concerned this would typically be delivered in a fixed format and not be directly editable. A user wanting to update an entry or correct an omission might typically be required to contact the webmaster. Typically there would be no provision for an individual user to alter the display format. From the point of view of this paper, two questions arise with a linked page of this kind. Firstly could users be given control over the content and styling of such information? For example could users decide that they preferred to sort the information by forename and omit the room number information and perhaps have it displayed in larger than normal font? Secondly could users update the information directly on the understanding that responsibility for the correctness of the information lies with the users themselves? These two issues are dealt with separately below.

3.1. Accessing and styling the shared data

The assumption is made that shared data is stored in database tables and that users should be given the opportunity to create a page based essentially on styling the results of a query of the database. What kind of interface would be appropriate for creating and editing such a page? With the cooperation of the webmaster in uploading the new page, a knowledgeable user could create a dynamic web page ‘by hand’ using a scripting language like PHP with embedded SQL commands. A solution that supposedly requires less intimate knowledge of scripting and database lookup would be to use the wizard facilities offered by sophisticated web page editors like Dreamweaver. However these wizards produce fully scripted pages (in PHP, ASP, etc). A more high-level approach presented here is to design a form-based editor which allows the users to select the data they are interested in, the ordering of the data and the styling. It is these selections, which capture the desired result in parameterised form, that are the outcome of the editing process. In use the parameters are supplied as a query string to a suitable script. Thus the URL for a link from the homepage takes the familiar CGI appearance

base_address ? search_path

where the base address is a fixed, generalised script which receives parameters from the ‘search_path’.The search path itself is a (potentially very long) sequence of parameters defining which tables and fields are to be selected, the ordering that is to be applied to the results, the styling of the results and the details of any headers, footers, etc. which complete the page.

The form-based editor breaks an SQL SELECT statement into sections, which can be roughly stated as

the FROM table at the heart of the query
    any JOINed tables with their JOIN ON details
a SELECTed list
    usually a collection of fields from the tables in question
the WHERE clause
    broken down into segments connected by AND/OR
    and subject to enclosure in parentheses
the ORDER BY clause
    with any number of fields used to specify the order

SQL practitioners will realise that the first two entries in the list above are the opposite way round to their appearance in a SELECT statement. They appear on the form in that order so that the user is guided to make a selection of a table before a field. This allows the menus for field selection to be populated with actual field names from the (already) selected tables.

In order not to make the form too limiting there are links at the bottom that cause the form to be redrawn with more or less fields to allow more/less JOIN tables, more/less selected fields, more/less terms in the WHERE clause, more/less fields in the ORDER BY part to be entered.

add/remove [- join +] [- select fields +] [- where +] [- order +]

There are radio buttons to choose a field for totalling. There is a check box to select between ALL and DISTINCT (with SQL meaning).

When the submit button is pressed the SELECT query is constructed from the data supplied by the form. For example each join clause is represented by three fields, a table and the left and right hand sides of an ON L=R clause. Thus the i'th JOIN is represented by three form elements named ti, jli, jri and these cause the phrase "LEFT JOIN value(ti) AS ti ON value(jli)=value(jri)" to be added to the SELECT statement. When the SELECT query has been fully formed it is presented to the database and the results displayed, styled as requested.

The basic styling is achieved by five elements which are shown below with simple but realistic examples of values.

heading	Staff Directory
pre	<table border="1" >
item	<tr><td>$<td><td>$</td></tr>
post	</table>
footer	<hr>

The HTML entered is concatenated in order but with the item element being repeated for each record selected from the database. Each '$' represents one field in the record. The output does not have to be styled as a table. For example it could be styled as a list using pre=
, item=
$,$, post= and as a simple paragraph by using pre=<p>, item=$,$
, post= . The display can be much more elaborate if required using stylesheets e.g. pre= .

Once the editor has been used to construct a successful search the full URL with all its parameters in the search path can viewed in a textarea. A button allows the URL to be added to the user's portal homepage as a new or replacement link. When the link is used the resulting page has styled results as requested but there is the option to redisplay the editing form at the bottom of the page, pre-filled with all the selections that defined the current query. This allows

the user at any time to tinker with the selection or styling and potentially update the link on their homepage.

Obviously an editor which requires a minimum of 20 parameters (typically ~50) to be filled in to create a dynamic page would be daunting if the user always had to start from scratch. But the idea is that the portal should offer 'default' versions of dynamic pages that the user might be interested in, so that the user's contribution is merely to 'tinker' with the parameters. So in the example quoted above the user may add the default staff directory link to their homepage and then using the editor make perhaps two or three minor alterations to obtain the version of the staff directory that really suits them.

Two fairly sophisticated styling options that have been found helpful are multi-column display and row spanning. The multi-column display is useful when the records naturally fill less than half of the screen width. Row span is useful when a column has repeated adjacent entries. In this case the entry is only given once and the span of the cell is extended downwards to include all the repeats.

3.2. Write access to the shared data

All of the description so far has been about selecting and styling data. Since this is a collaborative environment the view is taken that the users should be able to edit any data that they see is inaccurate. The overriding consideration driving this part of the design is that, when a user spots a piece of information that is incorrect or out-of-date, they can have an immediate interaction to fix the data. The understanding is that the user does this in a cooperative spirit to ensure the best data is available to all users. Now it is commonplace to provide editing facilities for database information but the facilities vary from specific editors to update selected information [e.g. a user updating their delivery details held by a supplier] to the ability to change any item of any table in a database (*e.g.* phpMyAdmin [14]).

Conventionally in web pages an edit of information that is on view requires a transfer to a different page on which the edit is completed. So the user starts on the data page, transfers to the edit page and then returns to the data page. However although this has become common practise it is not the most convenient or intuitive. A model interaction can be taken from a typical spreadsheet where data is updated 'on the page'. When spreadsheets were first introduced a click on a cell transferred the contents to a special area at the top of the sheet where the editing was completed. On clicking 'OK', or similar, the changed value was sent back to the originating cell. This produces an interaction which occurs all on one page. However spreadsheets have advanced so that altering a cell is now an in-place or in-line edit. Clicking on a cell turns it into the current cell for editing and its contents can be edited. Its contents are frozen again when the user presses enter or moves the focus to another cell.

Now the question arises as to whether this interaction can be achieved on a web-page. It seems that the answer is yes, almost. The main steps are as follows. Firstly any cell to be edited should be clickable, that is to say that it should have an onClick action. Secondly the data should now be editable. One way of achieving this is to have an <input> tag containing the same data (which was until now invisible) become visible and appear covering the exact place where the edit click was made. Finally when a button is pressed the alterations are sent to the database and the page redrawn.

It is this last part (the re-drawing) which provides the 'almost' in the answer above as to whether this web-based interaction can reproduce the spreadsheet interaction. For the situation where a small amount of data fits on a single screen, the redraw can appear as 'instantaneous' with a fast server and the user is hardly aware of the re-draw. For the situation where the page is

larger than the viewing screen, it is necessary to arrange that the redrawn page is scrolled to the same position as it was before the submit button was pressed. Even though this last step can cause a visible re-draw with a large amount of data and/or a slow server, it is still worth providing the first step of apparently editing ‘in-place’ to minimise the number of changes of focus required on the part of the user. If multiple changes are made before the alterations are saved then the speed of the re-draw is less of an issue.

Now the question arises as to whether all data drawn from a database table should be editable in this way. When using the editor to select and style the information to be displayed it may be convenient for the user to mark some data as not being editable. In other words given the freedom to edit anything, the user may still wish to limit the fields that they can edit in a particular display. This may be to prevent accidental editing or to increase efficiency by decreasing the number of fields which have the editing ‘overhead’. In order for the user to choose which fields should be editable a check box is provided on the editing form next to each (select) field. This allows the user to say whether the field is to be regarded as editable. If checked then clicking the field will allow editing.

In addition to the check box per field, there is another separate single check box which is used to switch editing off and on completely. Since pages that have editing enabled contain considerably more html text, it is useful to view certain pages normally with the edit ‘off’ for faster loading and only switching the edit ‘on’ for a specific occasion if required.

The edit is made possible by scripting the onClick event of a cell with a call of a javascript function, providing it with enough parameters to construct the SQL command to achieve the edit. Thus for example

```
onClick="update(tableA,field1,value1,field2,value2);"
```

might allow the following SQL command to be constructed

```
UPDATE tableA SET field1=value1 WHERE field2=value2
```

It is only possible for the system to create the required SQL command if there is a suitable key field defined for the table and further that the key field is included in the SELECT list of the main query. [Note that this does not necessarily mean that the field must be visibly on display because some fields can be hidden from view by using the style {display: none;}].

So far nothing has been mentioned about inserting or deleting whole records. These operations are made possible by extra Insert and Delete buttons alongside the Save (Submit) button. These buttons are restricted in use to the situation where all the data on view is from a single table (i.e a JOIN has not ben used).

3.3. Example - Supervision of project students

An example of a scenario where the expectation would be to edit on every visit to a page is keeping records of meetings, for example student project supervision. The page may well be built from three tables one giving basic details about a student, one linking supervisors to students, one linking students to comments. The selection would be those students being supervised by a particular member of staff. The display might be a four column display giving a picture of the student in column one, some details (name, programme, part, id) in column 2, the title of their project in column three and finally a comments field. It would be the comments field that would be editable (in-place of course).

Of course a system of parameterised editing as described here could never create all the pages that might be required. The present system has expanded to include many features over and above the initial implementation. For example one addition was the ability to add extra AND clauses one by one to the WHERE part of the query and then later on to choose to add an AND or OR clause and show priority using parentheses. However in the end there will always be the need for some ‘hand’ coded pages. It is important though that the in-line or in-place editing ideas should be carried over routinely into these special pages in order to continue to foster the notion of shared ownership.

3.4. Handling of in-line editing

It is only the standardisation of modern browsers that has made it possible to provide natural in-place editing. However the possibility seems to have largely passed by unnoticed. In-page editing is presented in Rees [15] and an example of web-based in-place editing, spreadsheet style has been found (Blueshoes [16]). Because there is such a great legacy of editing in the old style, web designers can be forgiven for believing that no other interaction method is possible.

The basic steps required are to have a normally hidden ‘layer’ that is placed exactly in front of a cell to be edited.

```
<input  id="IDX" class="..." type="text" value="V"
        style="position: absolute; z-index:2;
                left: Lpx; top: Tpx; width: Wpx; height: Hpx;
                visibility: hidden;" />
```

As introduced above an editable cell will have an onClick event programmed, and the script will be responsible for making the editForm visible using

```
document.getElementById('IDX').style.visibility = 'visible';
```

Two possible approaches are possible. In the first only one 'invisible' layer is available and it is dynamically and precisely moved to the site of an edit click. The value of the cell that has been clicked is transferred to the invisible layer, which is then made visible. The user can then edit the value in the normal way. In this method there is a submit and re-draw for every cell edited.

Using this approach arrangements must be made to discover the coordinates of the cell that was clicked for editing. The properties offsetTop, offsetLeft can be used to discover the position of the cell from its id attribute. The offsets returned by these properties are relative to the parent element so the page offsets are obtained by stepping up the document tree, summing each offset found, like this

```
offsetY = document.getElementById(id).offsetTop;
while( (id = id.offsetParent) != null)
        offsetY += id.offsetTop;
```

A second possibility is to have a hidden input element in front of every cell on display, pre-filled with the value of the corresponding cell. When the user clicks a cell, all that has to be done is to make the invisible input element become visible. The difference here is that several cells can be edited before a button is pressed to save the changes. The styles of the input elements (e.g. border red) can be made distinctive so that the user can see which cells have been edited. It is helpful to include a check (using onbeforeunload) if the user tries to navigate away from the page with unsaved changes.

4. CONCLUSIONS

This paper has presented the scenario of building an intranet for the use of a cooperating group who share responsibility for the accuracy of data that they create. The group of users is regarded

as being reasonably computer literate and would not readily accept arbitrary restrictions when more freedom could be offered if a little more programming effort was put into the system. Thus the homepage should be highly configurable and the access to data not restricted to pre-programmed reports. Further, the update access should not be restricted to requests to the webmaster.

An attempt has been made to design an intranet facility which escapes the common restrictions and which offers some novel features. Two designs for very flexible homepage editors have been presented. It turns out to be quite difficult to get the feel of these editors just right as they are quite complex, yet are not in constant use so they have to be very intuitive. An approach to handling the reading and writing of shared data has been presented. Once again the success revolves around getting the right feel of the editor involved. Building a new dynamic page from scratch is quite demanding. In practise, most users, most of the time, only need to modify the parameters defining an existing page which is a much simpler task. The approach to updating data is novel both from the point of view of allowing all users update access and in that the style of the interaction is in-line editing.

All the ideas presented in this paper have been implemented and tested at various times on the author's departmental intranet. The cooperating group have been his colleagues. In-place editing by all users has been implemented for certain categories of data such as the departmental directories, meetings schedule, module titles, module organisers, etc.

It is perhaps inevitable in a real-life situation that true cooperation and shared responsibility will rarely be found, if at all. However it is believed that more data could be cooperatively owned and managed than is currently the case, and it is the restricted features of current intranets that are to blame. Positive benefits can be expected from giving groups of users a shared responsibility of data. They care more about the quality and accuracy of the data and are more motivated to keep it up-to-date.

The key technology behind the implementation is the CSS and DOM standardisation by the W3C and the progressive adherence of the various browsers to the standards. Prior to this effort it was not really possible to make cross-browser, cross-platform implementations given the browser wars, the difficulty of writing browser detection and browser specific code.

REFERENCES

[1] myYahoo, http://my.yahoo.com

[2] myGoogle, http://www.google.com/ig

[3] Manber, U., Patel, A. and Robinson, J. 2000 “Experience with Personalization on Yahoo!”, *Communications of the ACM*, vol 43, no. 8, pp. 35-39, http://portal.acm.org/citation.cfm?doid=345124.345136

[4] Bellas, F., Fernandez, D. & Muino, A. 2004 “A flexible framework for engineering "my" portals”, *Proceedings of the 13th international conference on World Wide Web*, Pages: 234 - 243, http://portal.acm.org/citation.cfm?id=988704

[5] Wege C., “Portal Server Technology” 2002 *IEEE Internet Computing*, vol 6, issue 3, pp 73-77, http://portal.acm.org/citation.cfm?id=613707, http://csdl2.computer.org/persagen/DLAbsToc.jsp?resourcePath=/dl/mags/ic/&toc=comp/mags/ic/2002/03/w3toc.xml&DOI=10.1109/MIC.2002.1003134

[6] Perugini, S. & N. Ramakrishnan, N. 2003 “Personalizing Interactions with Information Systems”, *Advances in Computers*, volume 57: Information Repositories, pages 323-382, http://citeseer.ist.psu.edu/perugini02personalizing.html

[7] Priebe, T., & Pernul, G. 2003 “Towards Integrative Enterprise Knowledge Portals”, *Proc. of the Twelfth International Conference on Information and Knowledge Management (CIKM 2003)*, New Orleans, LA, USA, November 2003, http://citeseer.ist.psu.edu/priebe03towards.html

[8] Dignum V. 2004 “Personalised support for knowledge sharing”, *Proceedings of the conference on Dutch directions in HCI*, Amsterdam, Holland, published by ACM, http://portal.acm.org/citation.cfm?id=1005220.1005222

[9] Stone, R.G. 2001 “Building a Co-operative Evolutionary Intranet: Experience in a University Department” , *Software Quality Management* , Dawson, R., King, G., Ross, M. and Staples, G. (eds), British Computer Society, SQM 9 , Loughborough University, UK, pp 287-293, ISBN 1-902505-40-9, http://hdl.handle.net/2134/2633

[10] Mohaniyer 1998 *American History Quiz*, http://mohaniyer.com/americanhistoryie.htm (Internet Explorer only)

[11] Goodman, D. 1999 “Browser-Friendly Dragging in Dynamic HTML”, http://web.archive.org/web/20040603172427/http://developer.netscape.com/viewsource/goodman_drag/goodman_drag.html

[12] Dreamweaver, http://www.macromedia.com/software/dreamweaver/

[13] Cozens S. 2001 “Lightweight javascript drag-and-drop package dom-drag.js”, http://www.youngpup.net/2001/domdrag

[14] phpMyAdmin, http://sourceforge.net/projects/phpmyadmin/

[15] Rees, M. J. 2000 “Implementing Responsive Lightweight In-page Editing”, *Proceedings of AusWeb 2000*, Cairns, Australia, June, pp 134-142, http://ausweb.scu.edu.au/aw2k/papers/rees/paper.html

[16] Blueshoes, http://www.blueshoes.org/_bsJavascript/components/spreadsheet/examples/example1.html

(URLs accessed successfully 1/2/07)