Building multi-device, adaptive thin-client web user interfaces with Extended Java Server Pages

John Grundy and Wenjing Zou

Department of Computer Science, University of Auckland
Private Bag 92019, Auckland, New Zealand
john-g@cs.auckland.ac.nz

Abstract

We describe a new approach to providing adaptable thin client interfaces for web-based systems that allows a developer to specify a web-based interface using a device-independent mark-up language embedded in conventional Java Server Pages. At run-time this single interface description is used to automatically provide an interface for multiple web devices e.g. desk-top HTML and mobile WML-based systems, as well as highlight, hide or disable interface elements depending on the current user and user task. Our approach allows developers to more easily construct and maintain web-based user interfaces than other current approaches while utilising their existing server-side web components. We describe the software architecture of our system, its implementation using Java Server Page custom tag libraries, and an example application of our technique. We report our experiences using the technology to build three web-based applications and the results of two empirical studies of its effectiveness.

Keywords: adaptable user interfaces, web-based user interfaces, mobile device user interfaces, web-based software architectures, Java Server Pages

1. Introduction

Many web-based information systems require degrees of adaptation of the system's user interfaces to different client devices, users and user tasks [Van der Donckt et al 2001;Petrovski and Grundy, 2001]. This includes providing interfaces that will run on conventional web browsers, using Hyper-Text Mark-up Language (HTML), as well as wireless PDAs, mobile phones and pagers using Wireless Mark-up Language (WML) [Marsic, 2001a; Han et al 2000; Zarikas et al 2001]. In addition, adapting to different user and user tasks is required [Eisenstein and Puerta, 2000; Grunst et al 1995; Wing and Columb, 1996]. For example, hiding "Update" and "Delete" buttons if the user is a customer or if the user is a staff member doing an information retrieval-only task. Building such interfaces using current web-based systems implementation technologies is difficult, time-consuming and results in hard-to-maintain solutions.

Developers can use proxies that automatically convert e.g. HTML content to WML content for wireless devices [Marsic, 2001a; Han et al 2000; Van der Donckt et al 2001]. Typically these produce poor interfaces as the conversion is difficult for all but simple web interfaces. Some systems take XML-described interface content and transform it into different HTML or WML formats depending on the requesting device information [Marsic 2001a; Van der Donckt et al 2001]. The degree of adaptation supported is generally limited, however, and each interface type requires often complex, hard-to-maintain XSLT-based scripting. Intelligent and component-based user interfaces often support adaptation to different users and/or user tasks [Stephanidis, 2001; Grundy and Hosking 2001]. Most existing approaches only provide thick-client interfaces (i.e. that run in the client device, not the server), and most provide no device adaptation capabilities. Some recent proposals for multi-device user interfaces [Van der Donckt et al 2001; Han et al 2000; Marsic, 2001b] use generic, device-independent user interface descriptions. Most of these do not typically support user and task adaptation, however, and many are application-specific rather than general approaches. A number of approaches to model-driven web site engineering have been developed [Ceri et al, 2000; Bonifati et al 2000; Fraternali and Paolini, 2002]. Currently these do not support user task adaptation and their support for automated multi-device layout and navigation control is limited. These approaches typically fully-automate web site generation, and while valuable they replace rather than augment current development approaches.

We describe the Adaptable User Interface Technology (AUIT) architecture, a new approach to building adaptable, thin-client user interface solutions that aims to provide developers with a generic screen design language that augments current JSP (or ASP) web server implementations. Developers code an interface description using a set of device-independent XML tags to describe screen elements (labels, edit fields, radio buttons, check boxes, images, etc), interactors (buttons, menus, links, etc), and form layout (lines, tables and groups). These tags are device mark-up language independent i.e. not HTML or WML nor specific to a particular device screen size, colour support, network bandwidth etc. Tags can be annotated with information about the user or user task they are relevant to, and an action to take if not relevant (e.g. hide, disable or highlight). We have implemented AUIT using Java Server Pages, and our mark-up tags may be interspersed with dynamic Java content. At run-time these tags are transformed into HTML or WML mark-up and form composition, interactors and layout determined depending on the device, user and user task context.

The following section gives a motivating example for this work, a web-based job management system, and reviews current approaches used to build adaptable, web-based information system user interfaces. We then describe the architecture of our AUIT solution along with the key aspects of its design and implementation We give examples of using it to build parts of the job management system's user interfaces, including examples of device, user and user task adaptations manifested by these AUIT-implemented user interfaces. We discuss our development experiences with AUIT using it to build the user interfaces for three variants of commercial web-based systems and report results of two empirical studies of AUIT. We conclude with a summary of future research directions and the contributions of this research.

2. Motivation

Many organisations want to leverage the increasingly wide-spread access of their staff (and customers) to thin-client user interfaces on desktop, laptop and mobile (PDA, phone, pager etc) devices [Amoroso and Brancheau, 200; Varshney et al 2000]. Consider an organization building a job management system to co-ordinate staff work. This needs to provide a variety of functions allowing staff to create, assign, track and manage jobs within an organization. Users of the system include employees, managers and office management. Key employee tasks include login, job creation, status checking and assignment. In addition, managers and office management maintain department, position and employee data. Some of the key job management screens include creating jobs, viewing job details, viewing summaries of assigned jobs and assigning jobs to others. These interactions are outlined in the use case diagram in Figure 1.

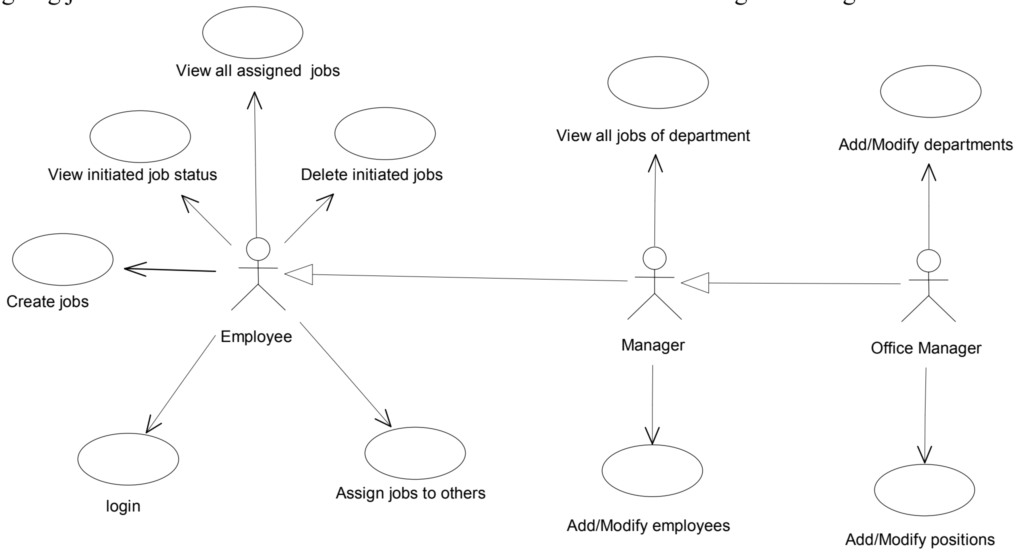
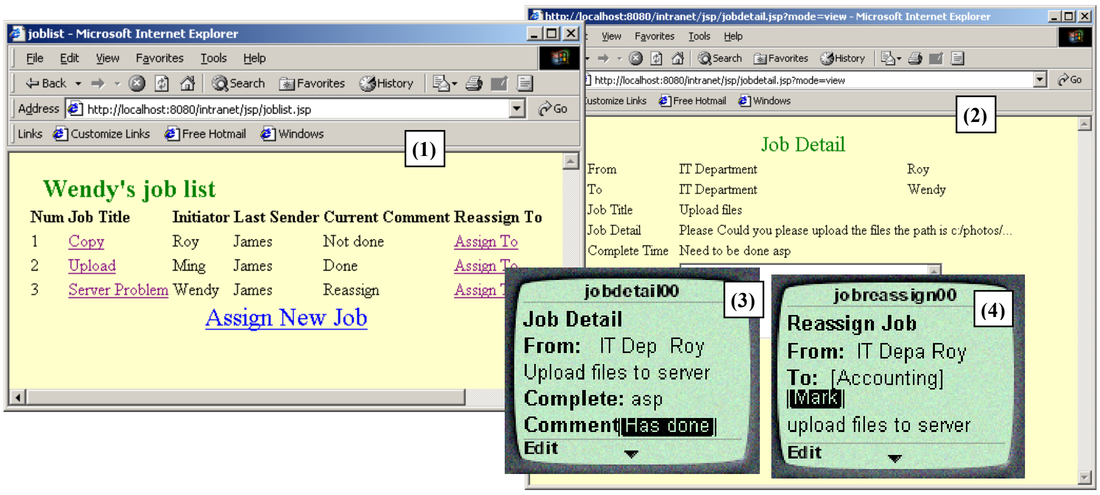


Figure 1. Example use cases in the job management system.

All of these user interfaces need to accessed over an intra-net using multi-device, thin-client interfaces i.e. web-based and mobile user interfaces. This approach makes the system platform and location-independent and enables staff to effectively co-ordinate their work no matter where they are.

Some of the thin-client, web-based user interfaces our job management information system needs to provide are illustrated in Figure 2. Many of these interfaces need to "adapt" to different users, user tasks and input/output web browser devices. For example, the job listing screens (1) for job managers and other employees are very similar, but management have additional buttons and information fields. Sometimes the job details screen (2) has buttons for modifying a job (when the owning user is doing job maintenance) but at other times not (when the owning user is doing job searches or analysis, or the user is not the job owner). Sometimes interfaces are accessed via desktop PC web browsers (1 and 2) and at other times the same interface is accessed via a WAP mobile phone, pager or wireless PDA browser (3 and 4), if the employee wants access job information when away from their desktop or unable to use their laptop.



joblist - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://localhost:8080/intranet/jsp/joblist.jsp Go
Links Customize Links Free Hotmail Windows
(1)
http://localhost:8080/intranet/jsp/jobdetail.jsp?mode=view - Microsoft Internet Explorer
View Favorites Tools Help
http://localhost:8080/intranet/jsp/jobdetail.jsp?mode=view
(2)
(3)
(4)

Wendy's job list

Num	Job Title	Initiator	Last Sender	Current Comment	Reassign To
1	Copy	Roy	James	Not done	Assign To
2	Upload	Ming	James	Done	Assign To
3	Server Problem	Wendy	James	Reassign	Assign T

Assign New Job

Job Detail

From	IT Department	Roy
To	IT Department	Wendy
Job Title	Upload files	
Job Detail	Please Could you please upload the files the path is c:/photos/...	
Complete Time	Need to be done asp	

jobdetail00
Job Detail
From: IT Dep Roy
Upload files to server
Complete: asp
Comment Has done
Edit

jobreassign00
Reassign Job
From: IT Depa Roy
To: [Accounting]
Mark
upload files to server
Edit

Figure 2. Some examples of adaptive Job Management information system screens.

To build user interfaces like the ones illustrated in Figure 2, we can use a number of approaches. We can build dedicated server-side web pages for each different combination of user, user task and device, using Java Server Pages, Active Server Pages, Servlets, PhPs, CGIs, ColdFusion and other technologies and tools [Marsic 2001a; Fields and Kolb, 2000; Evans and Rogers, 1997; Petrovski and Grundy, 2001]. This is currently the "standard" approach. This has the major problem of requiring a large number of interfaces to be developed and then maintained - for M different information system screens and N different user, user task and device combinations, we have to build and then maintain M*N screens. We can improve on this a little by adding conditional constructs to the screens for user and to some degree user task adaptations, reducing the total number of screens to build somewhat. However, for even small numbers of different users and user tasks we need adaptations for this approach makes screen implementation logic very complex and hard to maintain [van der Donckt, 2001; Grundy and Hosking 2001]. Each different device that may use the same screen still needs a dedicated server-side implementation [Fox et al, 1998; Marsic, 2001a] due to different device mark-up language, screen size, availability of fonts and colour and so on [van der Donckt et al, 2001].

Various approaches have been proposed or developed to allow different display devices to access a single server-side screen implementation. A specialised gateway can provide automatic translation of HTML content to WML content for WAP devices [Fox et al 1998, Palm, 2001]. This allows developers to ignore the device an interface will be rendered on, but has the major problem of many poor user interfaces being provided due to the fully-automated nature of the gateway. Often, as with Palm's Web Clipping approach, the translation cuts much of the content of the HTML document out to produce a simplified WML version, not always what the user requires.

Another common approach is to use an XML encoding of screen content and a set of transformation scripts to convert the XML encoded data into HTML and WML suitable for different devices [Han et al 2000; van der Donckt et al, 2001; Marsic, 2001]. For example, Oracle's Portal-to-go™ approach [Oracle Corp, 1999] allows device-specific transformations to be applied to XML-encoded data to produce device-tailored mark-up for rendering. Such approaches work reasonably well, but don't support user and task adaptation well and require complex transformation scripts that have limited ability to produce good user interfaces across all possible rendering devices. IBM's Transcoding™ [IBM Corp, 2002] provides for a set of transformations that can be applied to web content to support device and user preference adaptations. However a different transformation must be implemented for each device/user adaptation and it is unclear how well user task adaptation could be supported.

Another possibility is to use a database of screen descriptions and to convert, at run-time, this information into a suitable mark-up for the rendering device, possibly including suitable adaptations for the user and their current task [Fox et al 1998; Zarikas et al, 2001]. Another approach is to use conceptual or model-based web specification languages and tools, such as HDM, WebML and Autoweb [Ceri et al, 2000; Bonifati et al, 2000; Fraternali and Paolini, 2002]. These use device-independent specification techniques and typically generate multiple user interface implementations from these, one for each device and user. These approaches require sophisticated tool support to populate the database or generate web site implementations and are very different to most current server-side implementation technologies like JSPs, Servlets, ASPs and so on. Usually such systems must fully-generate web site server-side infrastructure and thus make it difficult for developers to reuse existing development components and approaches with these technologies.

Various approaches to building adaptive user interfaces have been used [Dewan and Sharma, 1999; Rossel, 1999; Eisenstein and Puerta 2000; Grundy and Hosking 2001]. To date most of these efforts have assumed the use of thick-client applications where client-side components perform adaptation to users and tasks but not different display devices and networks. The need to support user interface adaptation across different users, user tasks, display devices, and networks (local area, high reliability and bandwidth vs wide-area, low bandwidth and reliability [Rodden et al, 1998]) means a unified approach to supporting such adaptivity is desired by developers [van der Donckt, 2001; Marsic, 2001b; Zarikas et al, 2001; Han et al, 2000].

3. Our Approach

We have developed an approach to building adaptive, multi-device thin-client user interfaces for web-based applications that aims to augment rather than replace current server-side specification technologies like Java Server Pages and Active Server Pages. User interfaces are specified using a device-independent mark-up language describing screen elements and layout, along with any required dynamic content (currently using embedded Java code, or "scriptlets" [Fields and Kolb, 2000]). Screen element descriptions may include annotations indicating which user(s) and user task(s) the elements are relevant to. We call this Adaptive User Interface Technology (AUIT). Our web-based applications adopt the following four-tier software architecture, illustrated in Figure 3.

Clients can be desktop and laptop PCs running a standard web-browser, mobile PDAs running an HTML-based browser or WML-based browser, or mobile devices like pagers and WAP phones, providing very small screen WML-based displays. All of these devices connect to one or more web servers (the wireless ones via a wireless gateway) accessing a set of AUIT-implemented screens (i.e. web pages). The AUIT pages detect the client device type, remember the user associated with the web session, and track the user's current task (typically by which page(s) the current page has been accessed from). This information is used by the AUIT system to generate an appropriately adapted thin-client user interface for the user, their current task context and their display device characteristics. AUIT pages contain Java code scriptlets that can access JavaBeans holding data and performing form processing logic [Fields and Kolb, 2000]. These web server-hosted JavaBeans communicate with Enterprise JavaBeans which encapsulate business logic and data processing [Vogal, 1998]. The Enterprise JavaBeans make use of databases and provide an interface to legacy systems via CORBA and XML.

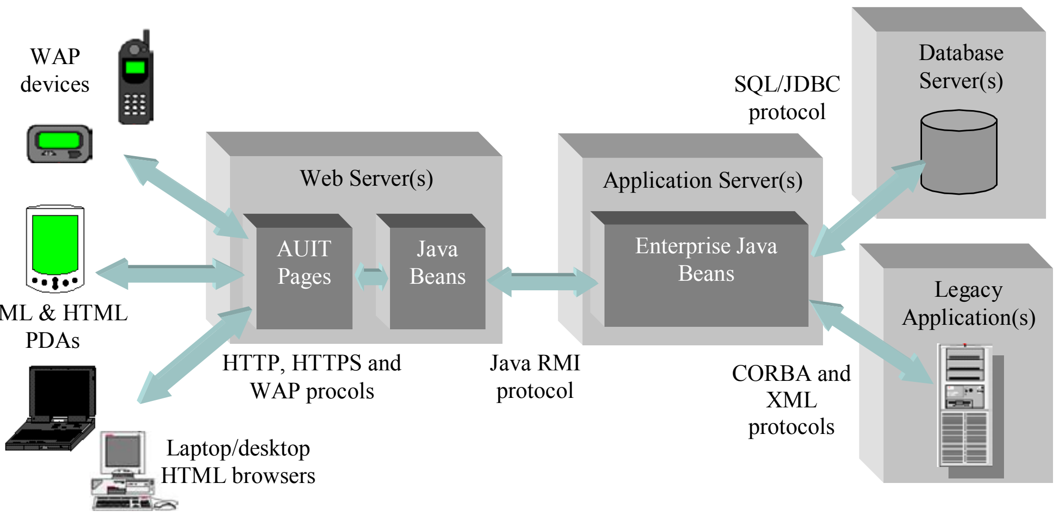


Figure 3. Our 4-tier web-based information system software architecture.

The AUIT pages are implemented by Java Server Pages (JSPs) that contain a special mark-up language independent of device-specific rendering mark-up languages like HTML and WML but that contain descriptions of screen elements, layout and user/task relevance, unlike typical data XML encodings. Developers implement their thin-client web screens using AUIT's special mark-up language, specifying in a device, user and task-independent way each screen for their application i.e. only M screens, despite the N combinations of user, user task and display device combinations possible for each screen. While AUIT interface descriptions share some commonalities with model-based web specification languages [Ceri et al, 2000; Fraternali and Paolini, 2002], they specify in one place a screen's elements, layout and user and task relevance.

An AUIT screen description encodes a layout grid (rows and columns) that contains screen elements or other layout grids. The layout grids are similar to Java AWT's GridBagLayout manager, where the screen is comprised of (possibly) different-sized rows and columns that contain screen elements (labels, text fields, radio buttons, check boxes, links, submit buttons, graphics, lines, and so on), as illustrated in Figure 4. Groups and screen elements can have priorities, user roles and user tasks that they are relevant to specified. This allows AUIT to automatically organise the interface for different-sized display devices into pages to fit the device and any user preferences. The structure of the screen description is thus a logical grouping of screen elements that is used to generate a physical mark-up language for different device, user role and user task combinations.

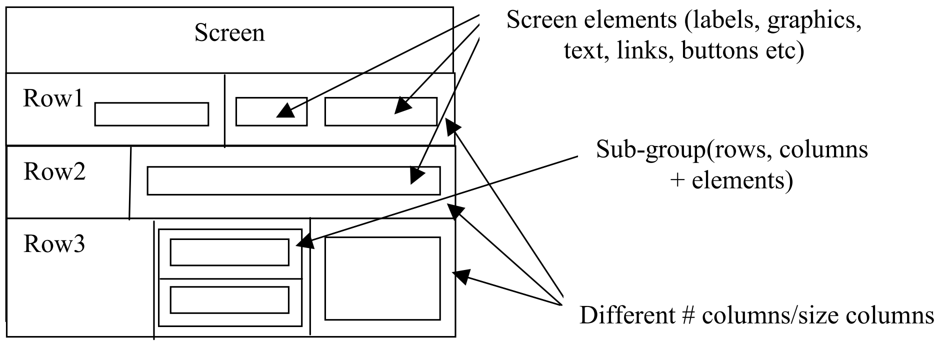

Screen
Row1
Row2
Row3
Screen elements (labels, graphics, text, links, buttons etc)
Sub-group(rows, columns + elements)
Different # columns/size columns

Figure 4. Basic AUIT screen description logical structure.

Unlike generic web mark-up languages and XML-encoded screen descriptions, AUIT screen descriptions include embedded server-side dynamic code. Embedded Java scriptlets currently provide this dynamic content for AUIT web pages and conventional JSP JavaBeans are used to provide data representation, form processing and application server access. When a user accesses an AUIT-implemented JSP page, the AUIT screen description is interpreted with appropriate user, user task and display device adaptations being made to produce a suitable thin-client interface. Using embedded dynamic content allows AUIT tags to make use of data as a device-specific screen description is generated. It also allows developers to use their existing server-side web components easily with AUIT screen mark-up to build adaptive web interfaces.

4. Design and Implementation

Java Server Pages are the Java 2 Enterprise Edition (J2EE) solution for building thin-client web applications, typically used for HTML-based interfaces but also usable for building WML-based interfaces for mobile display devices [Fields and Kolb, 2000, Vogal 1998]. To implement AUIT we have developed a set of device-independent screen element tags for use within JSPs that allow developers to specify their screens independent of user, task and display device. Note that we could implement AUIT in various ways, for example populate a AUIT-encoded screen description with data then transform it from its XML format into device-specific mark-up language, or extract AUIT screen descriptions from a database at run-time, generating device-specific mark-up from these. AUIT screen descriptions are typically lower-level than those of conceptual web specification languages e.g. WebML and HDM [Ceri et al, 2000; Bonifati et al, 2000] but we don't attempt to generate full web-side functionality from AUIT, rather interpret the custom AUIT tags and embedded Java scriptlets. AUIT descriptions have some similarities to some XML-based web screen encoding approaches, but again our focus is on providing JSP (and ASP) developers a device, user and user task adaptable mark-up language rather than requiring them to generate an XML encoding which is subsequently transformed for an output device.

Some of the AUIT tags and some of their properties are shown in Table 1, along with some typical mappings to HTML and WML mark-up tags. Some AUIT tag properties are not used by HTML or WML e.g. graphic, alternate short text, colour, font size, user role and task information, and so on. Reasonably complex HTML and WML interfaces can be generated from AUIT screen descriptions. This includes basic client-side scripting with variables and formulae – currently we generate JavaScript for HTML and WMLScript for WML display devices. AUIT tags are generally either layout control (screen, group, table, row, paragraph etc), page content (edit field, label, line, image etc), or control inter-page navigation (submit, link). Each AUIT tag has many properties the developer can specify, some mandatory and some optional. All tags have user and user task properties that list the users and user tasks the tag is relevant for. Screen tags have a specify task property allowing the screen to specify the a user's task context (this is passed onto linked pages by auit:link tags to set the linked form's user task. Grouped tags and table rows and columns have a "priority" indicating which grouped elements can be moved to linked screens for small-screen display devices and which must be shown. Table, row and column tags have minimum and maximum size properties, used when auto-laying out AUIT elements enclosed in table cells. Edit box, radio button, check boxes, list boxes and pop-up menus have a name and value, obtained from JavaBeans when displayed and that set JavaBean properties when the form is POSTed to the web server. Images have alternate text value and range of source files (typically .gif, .jpg and wireless bit map .wbm formats). Links and submit tags specify pages to go to and actions for the target JSP to perform.

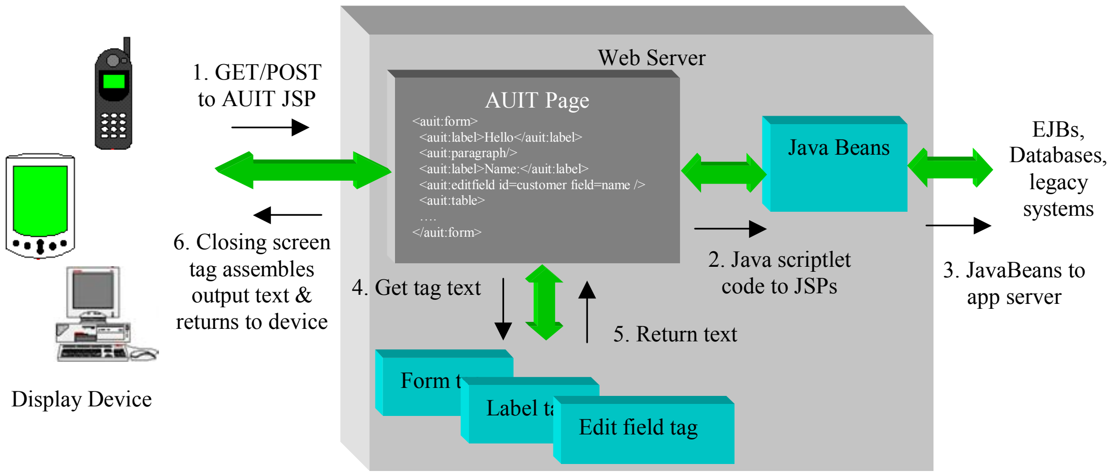
Users and user task relevance and priority can be associated with any AUIT tag (if with a group, then this applies to all elements of the group). We use a hierarchical role-based model to characterise users: a set of roles are defined for a system with some roles being generalisations of others. Specific users of the system are assigned one or more roles. User tasks are hierarchical and sequential i.e. a task can be broken down into sub-tasks, and tasks may be related in sequence, defining a basic task context network model. Any AUIT tag may be denoted as relevant or not relevant to one or more user roles, sub-roles, tasks or sub-tasks, and to a task that follows one or more other tasks. In addition, elements can be "prioritised" on a per-role basis i.e. which elements must be shown first, which can be moved to a sub-screen, which must always be shown to the user.

A developer writes an AUIT-encoded screen specification, which makes use of JavaBeans (basically Java classes) to process form input data and to access Enterprise JavaBeans, databases and legacy systems. At run-time the AUIT tags are processed by the JSPs using custom tag library classes we have written. When the JSP encounters an AUIT tag, it looks for a corresponding custom tag libray class which it invokes with tag properties. This custom tag class performs suitable adaptations and generates appropriate output text to be sent to the user's display device. Link and submit tags produce HTML or WML markups directing the display device to other pages or to perform a POST of input data to the web server as appropriate. Figure 5 outlines the way processing of AUIT tags is done. Note that dynamic content Java scriptlet code can be interspersed with AUIT tags.

AUIT Tag/Some Tag Properties	**Description**	**HTML**	**WML**
<auit:screen> • title, alternate • width, height • template • colour, bgcolour • font, lcolour	Encloses contents of whole screen. Title and short title alternate are specified. Can specify max width/height and AUIT appearance template to use. Default colours, fonts and link appearance can be specified.	<html>	<wml>, <card>
<auit:form> • action • method	Indicates an input form to process (POSTable). Specify processing action URL and processing method.	<form>	< do type=accept>
<auit:group> • width, height • rows, columns • priority • user, task	Groups related elements of screen. Group can have m rows, with each row 1 to n columns (may be different number). Number of rows can be dynamic i.e. determined from data iteration	-	-
<auit:table> • border, width • colour, bgcolour • rows, columns	Table (grid) with fixed number rows and columns. Can specify border width and colour, 3D or shaded border, fixed table rows/columns (if known)	<table>	<table>
<auit:row> • width, height • columns • user, task	Group or table row information. Can specify # columns, width and height row encloses. Can also restrict relevance of enclosed elements to specified user/task.	<tr>	<tr>
<auit:column> • width, height	Group or table column information	<td>	<td>
<auit:iterator> • bean, variable	Iterates over data structure elements. Uses JavaBean collection data structure.	-	-
<auit:paragraph>	Paragraph separator	<p>	<p>
<auit:line> • height, colour	Line break. Optional height (produces horizontal line)	 , <hr>	 , <hr>
<auit:heading> • level, colour, font • user, task	Heading level and text	<h1>, <h2> etc	Plain text
<auit:label> • colour, font • alternate, image • user, task	Label on form. Can have short form, image	Plain text	Plain text
<auit:textbox> • colour, font • user, task • script	Edit field description. Can define colour, fonts.	<input type=text>	<input type=text>
<auit:radio>	Radio button	<input type=radio>	<input type=radio>
<auit:select>	Popup menu item list	<select …>	<select …>
<auit:image> • source, alternate	Image placeholder, has alternate text (short and long forms)	<img src=…>	<img src=..>
<auit:link> • url, image • user, task	Hypertext link, has label or image	<a href=…>	<go href=…>
<auit:submit> • user, task • colour, image • url, script	Submit button/action (for form POSTing)	<input type=submit	<do><go href=…>
<% … %>	Embedded Java scriptlet code	-	-

Table 1. Some commonly-used AUIT screen element tags and some of their generated corresponding HTML and WML tags.

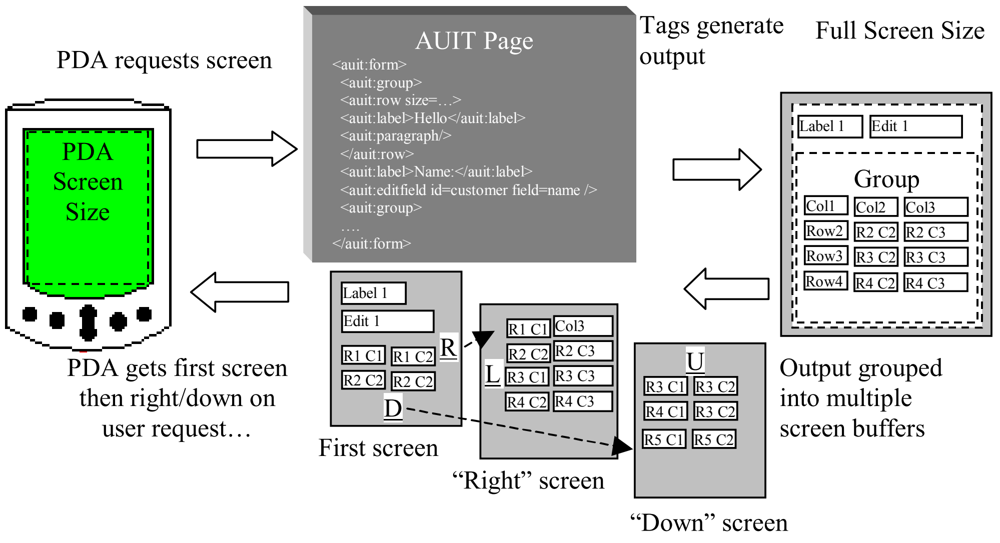
One of the more challenging things to support across multiple devices and in the presence of user and task adaptations being done (typically inappropriate screen elements being hidden) is providing a suitable screen layout for the display device. HTML browsers on desktop machines have rich layout support (using multi-layered tables), colour, a range of fonts and rich images. PDAs have similar layout capability, a narrower font range, and some have no colour and limited image display support. Mobile devices like pagers and WAP phones have very small screen size, one font size, typically no colour, and need low-bandwidth images. Hypertext links and form submission are quite different, using buttons, clickable text or images, or selectable text actions.



Web Server
AUIT Page
1. GET/POST to AUIT JSP
6. Closing screen tag assembles output text & returns to device
Display Device
Java Beans
EJBs, Databases, legacy systems
2. Java scriptlet code to JSPs
3. JavaBeans to app server
4. Get tag text
5. Return text
Form t
Label ta
Edit field tag

```
<auit:form>
  <auit:label>Hello</auit:label>
  <auit:paragraph/>
  <auit:label>Name:</auit:label>
  <auit:editfield id=customer field=name />
  <auit:table>
  ….
</auit:form>
```

Figure 5. How our JSP custom tag-implemented AUIT screen descriptions work.



```
<auit:form>
  <auit:group>
  <auit:row size=…>
  <auit:label>Hello</auit:label>
  <auit:paragraph/>
  </auit:row>
  <auit:label>Name:</auit:label>
  <auit:editfield id=customer field=name />
  <auit:group>
  ….
</auit:form>
```

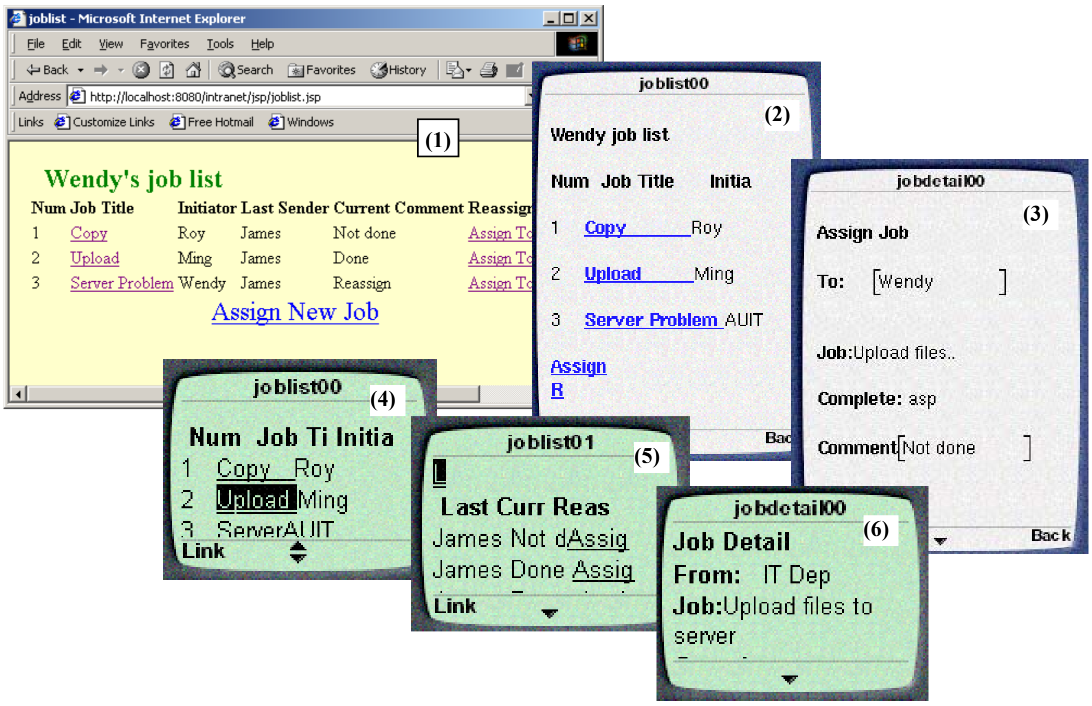
Figure 6. Example of screen splitting to adapt form layout for small-screen device display.

AUIT screens specifications enable users to use flexible groups to indicate a wide variety of logical screen element relationships and to imply desired physical display layouts. These allow our AUIT custom tags to perform automatic "splitting" of a single AUIT logical screen description into multiple physical screens when all items in a screen can not be sensibly displayed in one go on the display device. Group rows and columns are processed to generate physical mark-up for a device, and then are assembled to form a full physical screen. If some rows and/or columns will not fit the device screen, AUIT assembles a "first screen" (either top, left rows and columns that fit the device screen, or top-priority elements if these are specified). Remaining screen elements are grouped by the rows and columns and are formed into one or more "sub-screens", accessible from the main screen (and each other) via hypertext links. This re-organisation minimises the user needing to scroll horizontally and vertically on the device, producing a more easy to use interface across all devices. It also provides a physical interface with prioritised screen elements displayed. AUIT has a database of device characteristics (screen size, colour support and default font sizes etc), that are used by our screen splitting algorithm. Users can specify their own preferences for these different display devices characteristics, allowing for some user-specific adaptation support.

When processing an AUIT screen description, as each AUIT tag is processed it generates physical mark-up output text that is cached in a buffer. When group row or column text will over-fill the display device screen, text to the right and bottom over-filling the screen is moved to separate screens, linked by hypertext links and organised using the specified row/column groupings. An example of this process is outlined in Figure 6. Here a PDA requests a screen too big for it to display, so the AUIT tag output is grouped into multiple screens. The PDA gets the first screen to display and the user can click on the Right and Down links to get the other data. Some fields can be repeated in following screens (e.g. the first column of the table in this example) if the user needs to see them on each screen.

5. Job Management System Examples

We illustrate the use of our AUIT system used to build some adaptable, web-based job maintenance system interfaces as outlined in Section 2. Figure 7 shows examples of the job listing screen being displayed for the same user in a desktop web browser (1), mobile PDA device (2 and 3) and mobile WAP phone (4-6). The web browser can show all jobs (rows) and job details (columns). It can also use colour to highlight information and hypertext links. The PDA device can not show all job detail columns, and so additional job details are split across a set of horiztontal screens. The user accesses these additional details by using the hypertext links added to the sides of the screen. The WAP phone similarly can't display all columns and rows, and links are added to access these. In addition, the WAP phone doesn't provide the degree of mark-up the PDA and web browser can, so buttons and links and colour are not used. The user instead accesses other pages via a text-based menu listing.


joblist - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://localhost:8080/intranet/jsp/joblist.jsp
Links Customize Links Free Hotmail Windows
(1)
(2)
(3)
(4)
(5)
(6)

Wendy's job list

Num	Job Title	Initiator	Last Sender	Current Comment	Reassign
1	Copy	Roy	James	Not done	Assign To
2	Upload	Ming	James	Done	Assign To
3	Server Problem	Wendy	James	Reassign	Assign To

Assign New Job

joblist00

Wendy job list

Num	Job Title	Initia
1	Copy	Roy
2	Upload	Ming
3	Server Problem	AUIT

Assign R

Bac

jobdetail00

Assign Job

To: [Wendy]

Job:Upload files..

Complete: asp

Comment[Not done]

Back

joblist00

Num	Job Ti	Initia
1	Copy	Roy
2	Upload	Ming
3	Server	AUIT

Link

joblist01

Last	Curr	Reas
James	Not d	Assig
James	Done	Assig

Link

jobdetail00

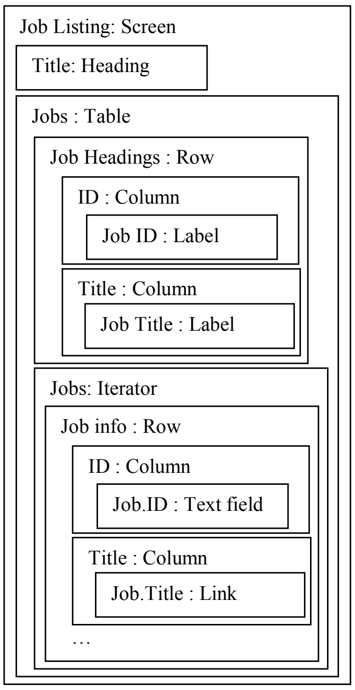
Job Detail

From: IT Dep

Job:Upload files to server

Figure 7. (a) Examples of job listing screen running on multiple devices.

Figure 8 (a) shows the logical structure of the job listing screen using the AUIT tags introduced in Section 4. The screen is comprised of a heading and list of jobs. The first table row displays column headings, the subsequent rows are generated by iterating over a list of job objects returned by a Java Bean. Figure 8 (b) shows part of the AUIT Java Server Page that specifies this interface. The first lines in the JSP indicate the custom tag library (AUIT) available and "JavaBean" components accessible by the page e.g. the job manger class provides access to the database of jobs. The screen tag sets up the current user, user task and device information obtained from the device and server session context for which the page is being run. The heading tag shows the user whose job list is being displayed. The table tag indicates a table and in this example one with a specified maximum width (in characters per row) and no displayed border. The first row shows headings per column, displayed as labels. The iterator tag loops, displaying each set of enclosed tags (each row) for every job assigned to the page user. The job list is obtained via the embedded Java code in the <% … %> tags. The column values for each row include labels (number, initiator, comment etc) and links (Job title, Assign To).



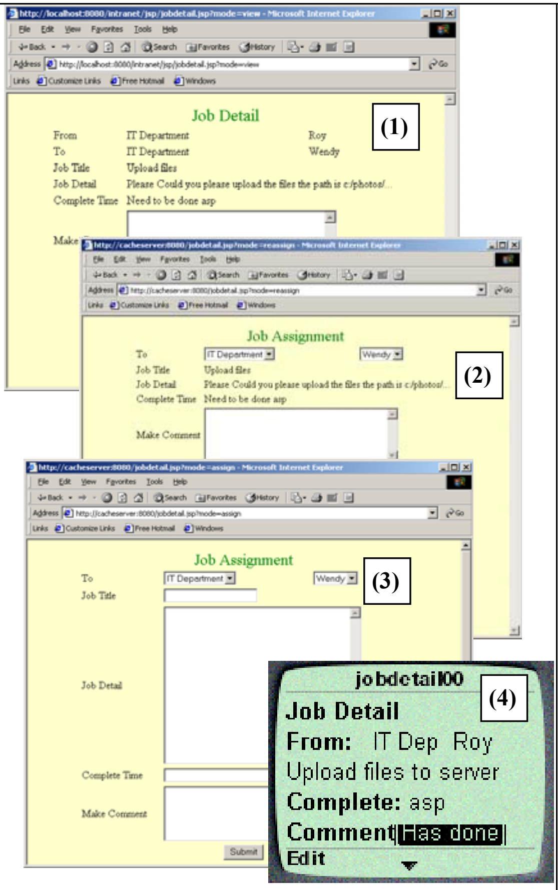
```
<%@ taglib uri="/auit" prefix="auit" %> // page directive to access AUIT tags
<jsp:useBean id='job_manager' class='jobs.JobManager /> // JavaBeans to use
…
<auit:screen name="job list"> // sets user/task/device information…
   <auit:heading level=2 value='<%= AUITUser.getUserName() %>'s Job List' />
   <auit:table width=60 border=0>
     <auit:row><auit:column><auit:label width=6 value='Num' /></auit:column>…
     <% jobs = job_manager.selectJobs(AUITUser.getUserName()); %>
     <auit:iterator name=job data=jobs %>
       <auit:row height=1>
         <auit:column><auit:label width=6 value=
                         '<% job.getJobNumber() %>' /></auit:column>
         <auit:column><auit:link width=20 name='<% job.getJobNumber() %>'
                         href='job_details.jsp?task=detail&job=
                                 <% job.getJobNumber() %>' /></auit:column>
         <auit:column><auit:label width=30 value=
                         '<% job.getInitiator() %>' /></auit:column>

         …
       </auit:row>
     </auit:iterator>
   </auit:table>
</auit:screen>
```

Figure 8. (a) Logical structure of and (b) some of the AUIT description of the job listing screen.

Figure 9 (a) shows examples of the job details screen in use for different users and user tasks. In (1), a non-owning employee has asked to view the job details. They are not able to modify nor assign other employees to this job. The same screen is presented to the job manager if they go to the job details screen in a "viewing" (as opposed to job maintenance) task context. In (2), the job manager has gone to the job details screen in task context "job re-assignment" from the job listing screen in Figure 7. They can assign employees to the job but not modify other job details. In (3) the job manager has gone to the screen to update a job's details and job details fields are now editable (a similar interface is presented when creating a new job). If an assigned employee accesses the job details in a job maintenance task context, only the comment and amount complete fields are editable, as shown in the WAP phone-displayed job details screen in (4).

Part of the AUIT specification of the job details screen is shown in Figure 9 (b). The screen encloses a form, which when the user fills out values for is posted to the web server for processing (done by the job_interface JavaBean component). The heading is task-dependent – if the user is viewing job details, the heading is different to if they are assigning or adding a job. The 'task' attribute of the two heading tags is used to determine which heading is shown. A table is used to achieve the layout. Some columns are common to all screens e.g. the left-hand side labels. Some rows are not shown for some screens e.g. the 'From' row is not shown if the user task is assign or new. Sometimes a different kind of form element is used e.g. if viewing a job, labels are used but when adding or assigning a job, some fields for the job have editable elements (text box, pop-up menu etc). If the user of the screen is not the job owner, then the delete button is not shown.



http://localhost:8080/intranet/jsp/jobdetail.jsp?mode=view - Microsoft Internet Explorer

Job Detail

From	IT Department	Roy
To	IT Department	Wendy
Job Title	Upload files	
Job Detail	Please Could you please upload the files the path is c:/photos/...	
Complete Time	Need to be done asp	
Make Comment		

http://cacheserver:8080/jobdetail.jsp?mode=reassign - Microsoft Internet Explorer

Job Assignment

To	IT Department	Wendy
Job Title	Upload files	
Job Detail	Please Could you please upload the files the path is c:/photos/...	
Complete Time	Need to be done asp	
Make Comment		

http://cacheserver:8080/jobdetail.jsp?mode=assign - Microsoft Internet Explorer

Job Assignment

To	IT Department	Wendy
Job Title		
Job Detail		
Complete Time		
Make Comment		

jobdetail00
Job Detail
From: IT Dep Roy
Upload files to server
Complete: asp
Comment Has done

```
<%@ taglib uri="/auit" prefix="auit" %>
<jsp:useBean id='job_interface' class='jobs.JobDetailsInterface />
<jsp:useBean id='job' class='jobs.JobData />
…
<% job_interface.processRequest(request, job); %>
<auit:screen name='job details'>
 <auit:form name='job details' action=job_details>
  <auit:centre>
   <auit:heading level=2 value='Job Detail' task='detail' />
   <auit:heading level=2 value='Job Assignment' task='assign|new' />
  </auit:centre>
  <auit:table width=60>
   <auit:row>
    <auit:column><auit:label value='From' /></auit:column>
    <auit:column task=detail><auit:label
         value='<% job.getUserFrom() %>'/></auit:column>
     …
    </auit:row>
    <auit:row>
     <auit:column><auit:label value='To' /></auit:column>
     <auit:column>
      <auit:label task=detail value='job.getUserTo() %></auit:column>
      <auit:select task='assign|new' name=to
            values='<% job.getAllUsers() %>' />…
     </auit:column>
     …
    </auit:row>
    …
  </auit:table>
  …
  <auit:submit user=<%= job.getUserFrom() %> name=action
              value=delete />
 </auit:form>
</auit:screen>
```

Figure 9. (a) Examples of adapted job details screen in use and (b) part of its AUIT description.

6. Experiences

To date we have built three substantial web-based applications with AUIT technology: the job management system, an on-line car sales E-commerce site, and an on-line collaborative travel planning system. Each of these systems have over a two dozen AUIT screens, JavaBean and EJB application server components, and database tables. We have also built, for other projects, "hard coded" versions of these systems using convention JSP technology - a commercial version of the on-line car retailer system, a commercial, in-house company job management system and a large prototype collaborative travel planning system. Each of these systems has JSPs specifically built for different users, user tasks and display devices. The AUIT systems all have less than a third of the screen specifications than these hard-coded systems. These AUIT screen specifications are easier to extend and maintain as new data and functions are added to the systems, as only a single specification needs modifying rather than up to a half a dozen for some screens in some of these hard-coded systems. Our AUIT technology allows developers to use all the usual JSP and Servlet functionality in conjunction with the AUIT adaptable tags. This means the expressive power for building dynamic web applications with AUIT-based JSPs is preserved, unlike when using XML-based translation approaches.

We have run two empirical evaluations of our AUIT-based systems, with a dozen end users comparing the AUIT and hard-coded system interfaces, and with half a dozen experienced, industry web-based information system developers comparing the use of AUIT technology to conventional JSP technology. End users in our studies using the job management and car site systems found the AUIT-implemented user interfaces to be as good from a usability perspective as the hard-coded ones. In fact, some found them better as they could change their device preferences and have the AUIT interfaces change to suit these, not possible with the hard-coded interface implementations. The software developers we had extend an existing JSP-implemented system by using AUIT to build two interfaces, both needing device, user and user task adaptations. They built two AUIT screen implementations and up to half a dozen conventional JSP implementations (generating HTML and WML respectively). These developers found AUIT to be straightforward to use and much more powerful and easier than the conventional JSP technology for building adaptable user interfaces. They found the range of functionality supported by AUIT tags to be large enough for most of their web-based user implementation needs.

We have tried to provide developers with a superset of screen specification tags and facilities enabling a wide range of both HTML- and WML-coded interfaces to be generated. However AUIT trades off the power of developers to tailor a web-based user interface to a specific device, user role and user task with ease of specification and maintenance of adaptable user interfaces. One consequence of specifying logical screen groups, elements and relevancies is that quite different physical screen layouts and interaction may well result depending on device, user and task accessing the interface. The interface the user sees is thus variable and may not be optimal i.e. usability is less for end users than hard-coded interfaces. An interesting and unintended consequence we found with AUIT user preferences is that users can specify dynamically their own preferred screen size limits, colour and image usage for different devices. AUIT then provides user-tailored interfaces using these preferences, an unintended but facility our end users found useful. We have found AUIT limited for highly image-oriented interfaces and interfaces using large numbers of embedded tables to provide very fine-tuned screen layout. AUIT is quite scalable in terms of number of users, imposing quite low overhead on the hosting server.

Our evaluation of AUIT have identified some areas for further research. The design of AUIT-based systems is radically different to user interface design of conventional web-based application interfaces. Designers need to work with logical structures like that illustrated in Figure 7 (b), than fixed-format layout as in conventional web user interface design. AUIT groups provide a reasonably flexible facility to layout screens, which work well for WML and moderately complex HTML interfaces. Very complex, fine-tuned HTML layout for desktop browsers is difficult to achieve with the current AUIT grouping components. Estimating the amount of room that rendered screen elements will take up, used by the screen splitting algorithm to move some information to linked screens, is difficult, as users may configure their device browsers with different default fonts and font sizes.

We are currently developing a new design method for adaptable web application user interfaces, along with a GUI specification tool that will generate AUIT implementations from these graphical designs. This will make it easier for developers to specify such systems interactively. We are continuing to enhance the layout control in AUIT grouping constructs to give developers more control over complex screen layout across display devices. Current task adaptation support is limited and we are extending this to allow developers to use more workflow-like information to support such adaptations. Extending user preference control and device characteristics, like network bandwidth, will allow further detailed specification of interface adaptation. AUIT could be applied to specifying thick-client adaptable user interfaces i.e. client-side user interface objects. Our aim to date however has been to focus on augmenting JSP server-side, thin-client adaptable user interface implementation, due to the limited client-side support in many small-screen devices.

7. Summary

We have developed a new approach for the development of adaptable, web-based information system user interfaces. This provides developers with a set of device-independent mark-up tags used to specify thin-client screen elements, element groupings, and user and user task annotations. We have implemented this with Java Server Page custom tag libraries, making our system fully compatible with current J2EE-based information system architectures. We have developed a novel automated approach for splitting too-large screens into parts for different display devices. We have developed several systems with our technology, all evaluated by end users and commercially deployable. Developers report they find our technology easier to use and more powerful for building and maintaining adaptable web-based user interfaces than other current approaches. End users report they find the adaptive interfaces suitable for their application tasks, and in some instances prefer them to hard-coded, device- and user-tailored implementations.

References

Amoroso, D.L. and Brancheau, J. (2001): Moving the Organization to Convergent Technologies: e-Business and Wireless, In Proceedings of the 34th Annual Hawaii International Conference on System Sciences, Maui, Hawaii, Jan 3-6 2001, IEEE CS Press.

Bonifati, A., Ceri, S., Fraternali, P., Maurino, A. (2000) Building multi-device, content-centric applications using WebML and the W3I3 Tool Suite, Proc. Conceptual Modelling for E-Business and the Web, LNCS 1921, pp. 64-75.

Ceri, S., Fraternali, P., Bongio, A. Web modelling Language (WebML): a modelling language for designing web sites, Computer Networks 33 (1-6), 2000.

Dewan, P. and Sharma, A. (1999): An experiment in inter-operating, heterogeneous collaborative systems, In Proceedings of the 1999 European Conference on Computer-Supported Co-operative Work, Kluwer, pp. 371-390.

Eisenstein, J. and Puerta, A. (2000): Adaptation in automated user-interface design, In Proceedings of the 2000 Conference on Intelligent User Interfaces, New Orleans, 9-12 January 2000, ACM Press, pp. 74-81.

Evans, E. and Rogers, D. (1997): Using Java Applets and CORBA for multi-user distributed applications, Internet Computing 1(3), 1997, IEEE CS Press.

Fields, D., Kolb, M. (2000): Web Development with Java Server Pages, Manning.

Fox, A., Gribble, S. Chawathe, Y., and Brewer, E. (1998): Adapting to Network and Client Variation Using Infrastructural Proxies: lessons and perspectives, IEEE Personal Communications **5** (4), August 1998, 10-19.

Fraternali, P. and Paolini, P. (2002) Model-driven development of web applications: the Autoweb system, to appear in ACM Transactions on Office Information Systems.

Grundy, J.C. and Hosking, J.G. (2001); Developing Adaptable User Interfaces for Component-based Systems, Interacting with Computers, Elsevier.

Grunst, G., Oppermann, R., Thomas, C. G. Adaptive and adaptable systems, In Hoschka, P. (ed.): *Computers As Assistants - A New Generation of Support Systems*. Hillsdale: Lawrence Erlbaum Associates, 1996. 29-46.

Han, R., Perret, V., and Naghshineh, M. (2000): WebSplitter: A unified XML framework for multi-device collaborative web browsing, In Proceedings of CSCW 2000, Philadelphia, Dec 2-6 2000, ACM Press.

IBM Corp, IBM Transcoding™ White Paper, http://www.research.ibm.com/networked_data_systems/transcoding/transcodef.pdf.

Marsic, I. (2001a): Adaptive Collaboration for Wired and Wireless Platforms, IEEE Internet Computing July/August 2001, 26-35.

Marsic, I. (2001b): An architecture for heterogeneous groupware applications, In Proceedings of the International Conference on Software Engineering, May 2001, IEEE CS Press, pp. 475-484.

Oracle Corp, Oracle Portal-to-go™ White Paper, October 1999, http://www.alentus.com/library/oracle/wp_portal.pdf.

Palm Corp. (2001): Web Clipping services, www.palm.com.

Petrovski, A. and Grundy, J.C. (2001): Web-enabling an Integrated Health Informatics System, In Proceedings of the 7th International Conference on Object-oriented Information Systems, Calgary, Canada, August 26-29 2001, Lecture Notes in Computer Science.

Rodden, T., Chervest, K., Davies, N. and Dix, A. (1998): Exploiting context in HIC Design for Mobile Systems, In Proceedings of the first Workshop on Human Computer Interaction with Mobile Devices.

Rossel M. (1999): Adaptive support: the Intelligent Tour Guide. 1999 International Conference on Intelligent User Interfaces. ACM. 1999, New York, NY, USA.

Stephanidis, C. (2001): Concept of Unified User Interfaces, In User Interfaces for All - Concepts, Methods and Tools, Laurence Erlbaum Associates, pp. 371-388.

Van der Donckt, J., Limbourg, Q., Florins, M., Oger, F., and Macq, B. (2001): Synchronised, model-based design of multiple user interfaces, In Proceedings of the 2001 Workshop on Multiple User Interfaces over the Internet.

Varshney, U. Vetter, R.J. and Kalakota, R. (2000): Mobile Commerce: A New Frontier, Computer **33** (10), IEEE Press.

Vogal, A.: (1998): CORBA and Enterprise Java Beans-based Electronic Commerce, International Workshop on Component-based Electronic Commerce, Fisher Center for Management & Information Technology, UC Berkeley.

Wing, H. and Colomb RM. (1996): Behaviour sharing in adaptable user interfaces. Proceedings Sixth Australian Conference on Computer-Human Interaction, IEEE CS Press, 1996, Los Alamitos, CA, USA, pp.197-204.

Zarikas, V., Papatzanis, G., and Stephanidis, C. (2001): An architecture for a self-adapting information system for tourists, In Proceedings of the 2001 Workshop on Multiple User Interfaces over the Internet.