An environment for developing adaptive, multi-device user interfaces

John Grundy[1,2] and Biao Yang[2]
Department of Electrical and Electronic Engineering[1] and Department of Computer Science[2]
University of Auckland, Private Bag 92019, Auckland, New Zealand
john-g@cs.auckland.ac.nz

Abstract

There is a growing demand for the development of multi-device, adaptive user interfaces – interfaces that will run on and adapt to the characteristics of multiple display devices and networks as well as multiple users and user tasks. We describe a design and implementation environment for the development of such interfaces. This tool allows developers to specify their desired interfaces using an abstract set of screen element and layout constructs. It then generates a Java Server Page implementation using a custom tag library that realises a multi-device, adaptive interface. We compare and contrast our approach to other techniques and describe our experiences using it.

Keywords: multi-device user interfaces, adaptive user interfaces, user interface design tools, mobile user interfaces, thin-client user interfaces.

1 Introduction

Many researchers and practitioners have identified the need for adaptive, multi-device user interfaces (Amoroso, and Brancheau, 2001; Grundy and Zou, 2002; Han et al, 2000; Van der Donckt et al, 2001). These systems provide user interfaces that can be used across platforms and/or display devices e.g. an interface that will run on both a conventional desktop web browser but also on a PDA or mobile phone device. This allows developers to design and implement a single interface that may be run and provide a userful interface on many different devices (some whose characteristics may be unknown to the developers). In addition, similar techniques can be employed to have the user interface adapt to the characteristics of different users and user tasks. For example, an Update button is hidden if the user is not sufficiently privileged or the user is performing a search vs a data maintenance task.

However, many challenges present when developing such interfaces (Grundy and Zou, 2002; Marsic, 2001b; Stephanidis, 2001). Many commercial "web clipping", portal and interface transformation technologies apply various filters to information to generate interfaces for particular display devices and users (IBM Corp, 2001; Oracle Corp, 1999; Palm Corp., 2001). Some systems provide server-side implementations of interfaces that at run-time consider the device and user characteristics and produce context-dependent interfaces (Bonifati et al, 2000; Han et al, 2000; Van der Donckt et al, 2001). Some basic design tools have been developed that support typically simple interface element and layout specification and generate hard-coded, device-specific implementations (Ceri et al 2000; Fraternali and Paolini, 2002).

We describe a design environment for multi-device, adaptive user interfaces we have developed. Currently this supports the generation of a single run-time adaptive interface implementation for thin-client user interfaces (HTML and WML) for desktop, PDA and mobile phone web browsers. Our tool provides the designer with several views of interface designs and implementation, including structure and abstract screen layout. All design views are kept consistent under change. Currently Adaptive User Interface Technology implementations are generated from an XML encoded of an interface design – these are extended Java Server Page implementations using a custom tag library to provide run-time device, user and user task adaptive features.

We begin by a motivating example for our work, then review related approaches. We outline our design environment and illustrate its support for designing adaptive interfaces using a single model presented in several ways. We illustrate integrated user interface testing and discuss the design and implementation of our tool. We conclude by summarising our experience to date designing and building adaptive user interfaces and some of our future research directions.

2 Motivation

Consider an on-line system for the car sales industry. The purpose of this system is to provide customers a multiple dealer on-line search and messaging facility, and car dealers a shared site for advertising their products. Some of the main use cases of this system are summarised in Figure 1 (a): customers view stock by dealers; search for cars with specified characteristics, and message dealers to ask questions/book cars for viewing etc. Dealers update stock and communicate with customers. Sales staff maintain various data. The interfaces to support these use cases can be implemented by HTML-based web pages and viewed by conventional desktop web browsers (many such car sales sites now exist). However, both dealers and some customers may wish to access the system away from their desktop PCs e.g. on a wireless PDA or mobile phone. Some of the user interfaces we have implemented for such an application are illustrated in Figure 1 (b) – 1 and 2 are desktop browser-viewed car search and results display; 3 and 4 are wireless PDA-viewed equivalents and 5 is a search being requested from a mobile phone.

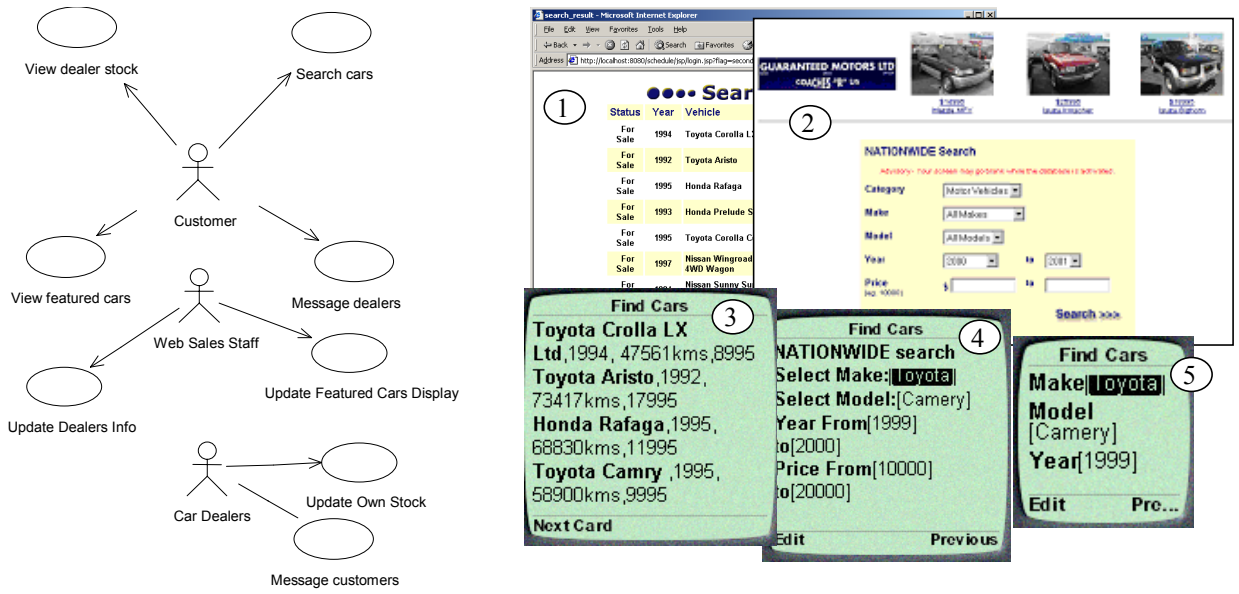


Figure 1. (a) Use case model for an on-line car site; and (b) examples of multi-device interfaces.

A variety of approaches have been developed to support such adaptive user interface development. A number of thick-client approaches using adaptive software components have been implemented (Dewan and Sharma, 1999; Eisenstein and Puerta, 2000; Grundy and Hosking, 2001; Morch, 1998). These approaches provide good user and task adaptation support, but typically do not run on anything but desktop machines i.e. are unsuitable for small-screen, mobile display devices.

Due to the increase in popularity of these devices, there has become a large demand for conventional web-based interfaces to be made available for them. Typical approaches are to hard-code interfaces specifically for such devices or to provide portals or "clipping" and transformation services that attempt to transform interface descriptions intended for standard web browsers on desktop machines to small-screen displays (IBM Corp, 2001; Oracle Corp., 1999; Palm Corp., 2001). The major problem of these approaches is lack of information in the transformation engine about the meaning and purpose of various interface elements, resulting in often poor transformed user interfaces.

A variety of technologies have been developed to support server-side adaptive interfaces for thin-client devices (Ceri et al 2000; Grundy and Zou, 2002; Han et al, 2000; Marsic, 2001b; Van der Donckt et al, 2001; Zarikas et al, 2001). Most of these approaches adopt XML-based encoding of data and transform this into a variety of target user interface implementations. Many do not support user or task adaptation and essentially provide statically generated interfaces for a limited range of devices known at generation-time.

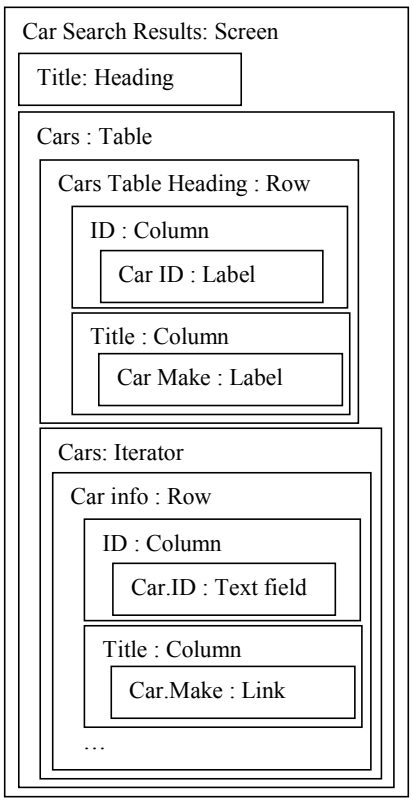
Some design tools have been developed based on generic description languages for web-based user interfaces (Bonifati et al, 2000; Fraternali and Paolini, 2002). They support specifying encodings of interface descriptions for the generation of interface implementations. The encodings are typically specified in an abstract manner not corresponding to eventual interface appearance and behaviour.

3 Our Approach

We have developed a technology for implementing run-time adaptable multi-device user interfaces using a set of custom tag libraries for Java Server Pages (JSPs), that we call Adaptive User Interface Technology (AUIT) (Grundy and Zou, 2002). Developers implement these extended JSPs describing interface elements and layout information with generic, device-independent mark-up tags. They may also indicate the users (by roles) and user tasks (by role-assigned tasks) for which some tags are (in)appropriate. At run-time when an AUIT JSP page is accessed, it determines the characteristics of the requesting display device (client), user of that device, and the user's current task.

When formatting mark-up to send back to the device for display it uses this information to provide suitable mark-up (e.g. HTML or WML), layout (e.g. multiple screens/cards for small-screen devices), interaction (e.g. buttons or command list items), adornment (e.g. available fonts only; no colour if black-and-white device), graphic (e.g. high-res GIF or low-res, monotone WBMP) and hides inappropriate screen elements for the user and user's task.

Figure 2 shows an part of an AUIT JSP on the right hand side. The user must specify a large number of tags and tag parameter values in order for AUIT tag library classes to appropriate adapt interfaces at run-time. When designing these implementations, we often use an abstract structure, as shown on the left, to guide the specification of the hierarchical tag structures.




```
<%@ taglib uri="/auit" prefix="auit" %> // page directive to access AUIT tags
<jsp:useBean id='car_manager' class='car.CarManager /> // JavaBeans to use
…
<auit:screen name="car search result"> // sets user/task/device information…
   <auit:heading level=2 value='Car Search Result' />
   <auit:table width=60 border=0>
    <auit:row><auit:column><auit:label width=6
                        value='Num' /></auit:column>…
    <% cars = car_manager.selectCars(…); %>
    <auit:iterator name=car data=cars %>
     <auit:row height=1>
      <auit:column><auit:label width=6 value=
                        '<% car.getCarID() %>' /></auit:column>
      <auit:column><auit:link width=20 name='<% car.getCarID() %>'
                        href='car_details.jsp?task=detail&car=
                              <% car.getCarID() %>' /></auit:column>
      <auit:column><auit:label width=30 value=
                        '<% car.getMake() %>' /></auit:column>
      …
     </auit:row>
    </auit:iterator>
   </auit:table>
</auit:screen>
```

Figure 2. Adaptive User Interface Technology example.

Hand-coding these AUIT implementations is time-consuming and error-prone, just like hand-coding concentional JSP implementations and HTML pages (Evans and Rogers, 1997). To enable developers to much more easily design, test and implement such adaptive interfaces a design tool is necessary. This might not only be able to generate AUIT implementations from the designs but also other targets e.g. conventional JSP or servlets, XSLT scripts or WebML (Ceri et al 2000; Fields and Kolb, 2000).

Key requirements for such a GUI design tool are rather different than for conventional GUI design tools like those of MS Access™, JBuilder™ or VisualBasic™. These include:

- Ability to design adaptive user interfaces where the interface elements and layout may change depending on device, user and user task accessing the interface at run-time. A designer is thus working with a template for multiple target user interfaces rather than just one.
- Use of various visualisations of the interface design and implementation including hierarchical tree-structure, screen layout and interface implementation target source language. Tree structured visualisations are useful due to the hierarchical nature of the interface descriptions (as shown in Figure 2 (b)). A screen layout view is similar to that of conventional user interface builder tools but would present an "abstract" layout of the screen, like those we use when designing AUIT interfaces (as shown in Figure 2 (a)).
- Ability to view an interface design as it would appear in target devices. Ideally support for interacting with the interface would also be provided to the designer.
- Multiple implementation languages for the interface design supported. As indicated in the previous section, a range of implementation techniques exist to realise adaptive user interfaces, not only our AUIT, and generating different implementations of the same adaptive user interface design may be required for use in different situations e.g. for compatibility with other interface implementations.

4 Tool Architecture

We have developed a prototype design tool for the specification of adaptive user interface designs. The architecture of this tool is outlined in Figure 3. Designers can view and edit an interface design using a screen layout, hierarchical logical or textual view. All views share a common design model. This model is encoded in an XML format for storage or to enable generation of target interface implementations. A range of interface implementations are possible – currently we generate JSP pages with AUIT custom tag library mark-up.

Figure 4 illustrates the main components of our design tool and their major inter-relationships. A Swing based user interface allows designers to view and edit logical views (tree structure visualisation), textual views (indented XML data model), and abstract screen layout (tiled mutli-sized panels and interface elements). Interface designs can be visualised and interacted with using embedded Internet Explorer (web browser) and Nokia Mobile Toolkit (PDA and mobile simulator) views.

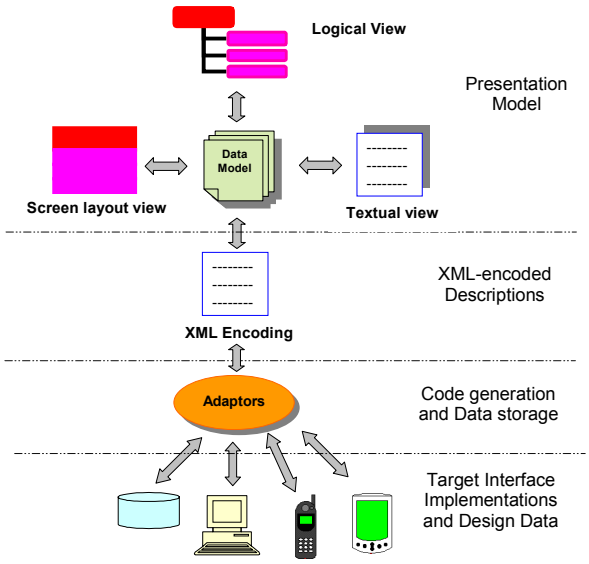


Figure 3. Adaptive GUI design tool architecture.

A shared screen design data structure is used by all design views. We have based this logical design model on the AUIT custom tag library components we have developed for realising adaptive user interfaces. However we have further-generalised these components to allow a wide range of user interface elements, interactors and layout constraints to be specified, along with user and task model information, independent of any specific target implementation technology. This data structure is converted into an XML model for storage and for target implementation code generators. Java scriptlet tags (server-side interface functionality) may be specified by designers and JavaBeans referenced, with both embedded in the generated target implementations of designs. These allow designers to fully-implement their interfaces and associated server-side functionality within our tool.

Our XML model is used to generate AUIT implementations (one per interface design), but we have also prototyped code generation for standard JSP and servlet implementations (one per device/user/task combination). We use a set of XSLT transformation scripts to convert our XML-encoded logical screen design into these target implementations.

In order to display running designs as they would appear in target devices with example user role and user task, we use Internet Explorer 6 (IE6) and the Nokia Mobile Toolkit. A Tomcat JSP engine is used to run target implementations and display them in IE6 and Nokia toolkit windows. Active-X controls are used to incorporate these inside our design environment's display panels. The designer may interact with these interfaces and depending on the amount so far specified, embedded Java scriptlets and hypertext links if specified allow sophisticated functionality and navigation to be tested.

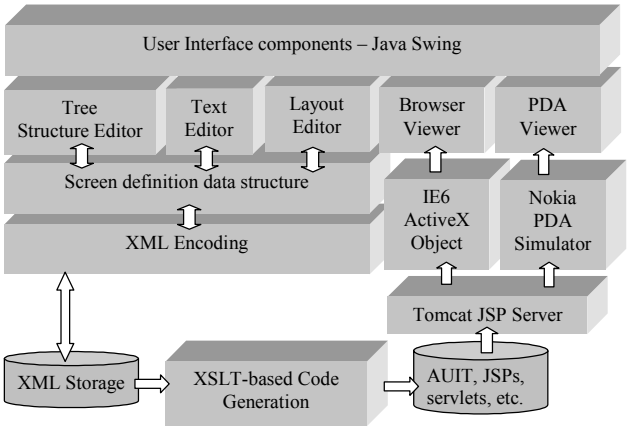


Figure 4. Tool components.

5 Design and Implementation

Figure 5 shows a screen dump from our design environment. The environment provides a Swing-based user interface with menu and tool bars (1) for design document management. The set of views of a design (2) include logical structure view (tree hierarchy), screen layout, source (textual) view and previews in browser and PDA/mobile phone simulator. When editing logical structure and screen layout views a tool bar with screen elements, layout control, navigation and scriptlets is available (3). Each view displays information in its particular format (4) and provides appropriate editing support e.g. tree manipulation, panel tiling, and text editing. Properties for elements can be set by direct manipulation in some cases, and in dialogues in others (5).

Consider designing adaptable interfaces for the on-line car site illustrated in Figure 1. Interfaces required include a login screen, a dealer catalogue, dealer search and dealer information, car search and detailed information display, and data update screens – cars, dealers, makes and models, and so on. Each of these interfaces can potentially be accessed by various display devices (desktop browsers, PDAs, mobile phones etc), and some can be used by different kinds of users or during different user tasks (e.g. customers vs dealership staff; searching task vs data maintenance task).

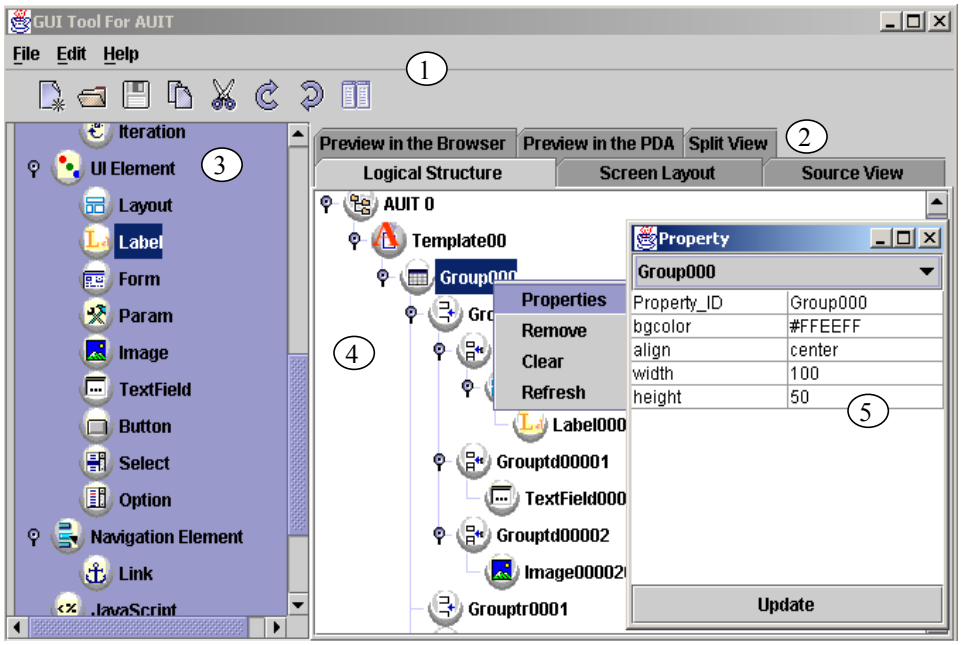


Figure 5. Overview of the design tool user interface.

A designer can choose one of the three editing views to construct an interface – the logical structure view, which provides a tree structure editor; a screen layout view, which provides a tiled view; and a "source view" which currently provides an indented XML editor using an XML encoding of the shared data model. Designers may switch between views at any stage for the same interface design, and changes to one view are automatically propagated to the other two views.

In Figure 6 (1) the designer is viewing and editing a design for the login screen with the logical structure editor. This presents the design as an editable tree structure, allowing the designer to browse complex interfaces and to perform tree editing operations to modify their design. The choice of a tree visualisation is due to the very hierarchical nature of AUIT (and other) screen implementation technologies – feedback from previous users of AUIT indicated that they felt a tree-editor would be an effective way to visualise and modify their adaptive interface designs (Grundy and Zou, 2002). The designer can select (2) tree nodes, corresponding to screen elements, layout constraints, navigation elements or Java scriptlets and JavaBean references. The detailed properties of different element types can be edited in a dialogue (3), and changes to the tree structure, changing the logical structure of the interface design, are affected using the tool-bar (4).

Designers may also choose to view and/or edit an auto-indented textual view of their interface design (5). This is sometimes a more convenient form to interact with, particularly when specifying Java scriptlets and JavaBean expressions, which need to be done textually. For the expert designer it is also quicker to edit, particularly when wanting to change properties associated with many screen elements at a time.

Currently we use an AUIT-based XML rendering to display the screen design textually, which developers may then modify with text editing operations (6). Any changes made to this textual representation are parsed back into XML and then reflected back into the shared data model, and from there in the other design views. We chose to use an AUIT-based XML model to display the interface design in a textual form due to our familiarity with this adaptive user interface implementation technology. It is also easy to both render the shared design model in this form and to translate editing operations on this XML structure back into changes on the shared data model.

AUIT provides a GridBag-style layout control mechanism to support the implementation of complex interface layouts. This can be translated into HTML Table tags or into WML layouts using multiple cards and tabbed field separation.

An important characteristic of AUIT-implemented interfaces is that the grid structure can be used at run-time to divide a large interface into multiple, logically sensible smaller interfaces for display on small-screen devices (Grundy and Zou, 2002). Similarly, if generating hard-coded JSP or servlet implementations for a particular small-screen display device, this splitting can be done at code generation time rather than at run-time as with AUIT. A further design view available is the screen layout view which allows the designer to see and directly manipulate this grid-based layout structure. This provides a tiled interface, as shown in **Figure 7**, where the user can see the relative positions and groupings of screen elements based on the grid model. The screen layout view allows the designer to see and edit this grid layout structure and to manipulate screen elements embedded in each grid cell. Note that this direct manipulation design tool is very different from MS Access™, JBuilder™ and Visual Basic™ drag-and-drop GUI design tools.

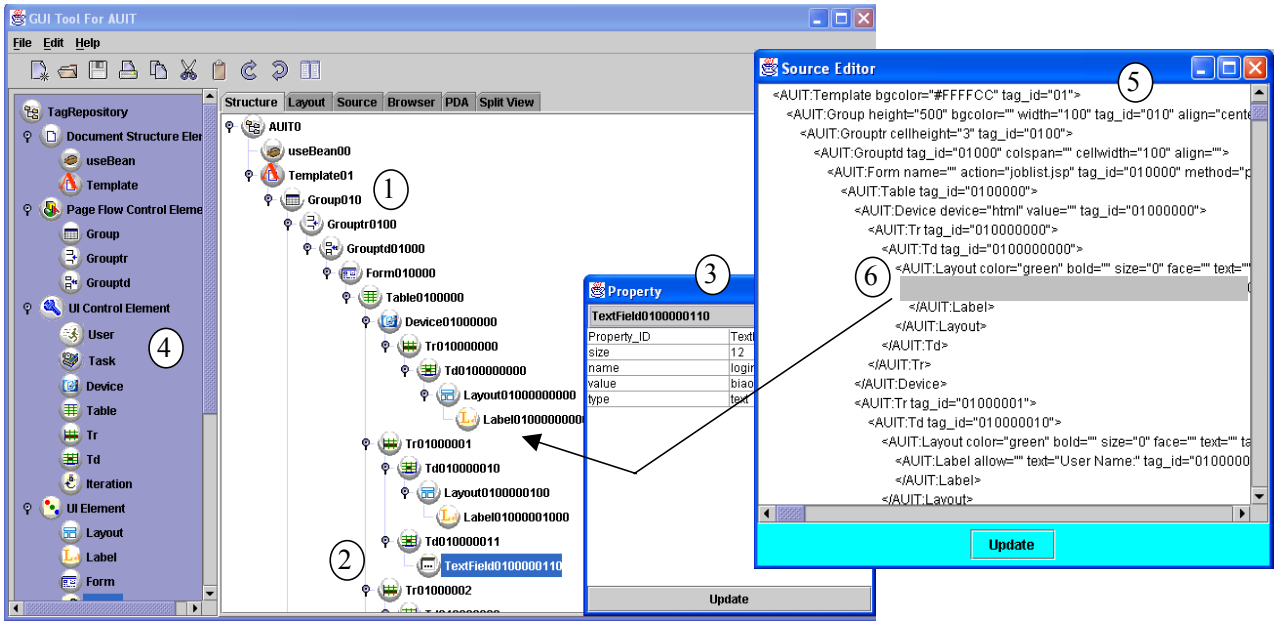


Figure 6. Examples of using of the structure editor for GUI design.

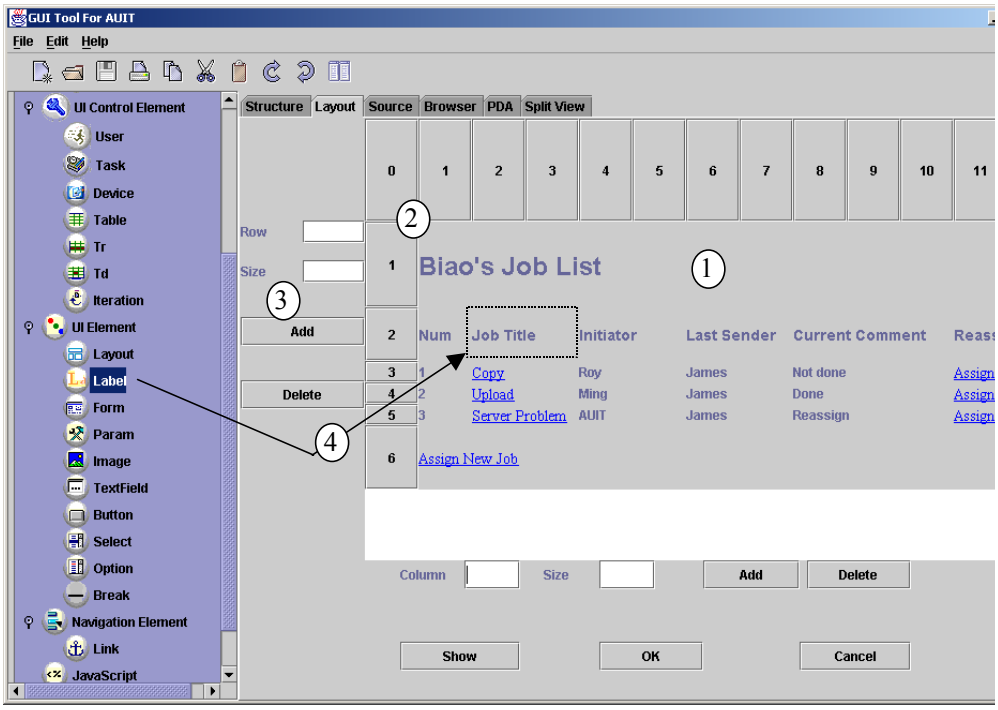


Figure 7. Example of using the screen layout view.

Unlike these tools, however, the screen layout view does not show a what-you-see-is-what-you-get view of the resultant implemented GUI interface, but a rendering of its entire logical structure using a tiled grid view. As mentioned above, when running the interface for a particular display device it may be split into multiple parts using the grid specified, in order to fit the interface to in parts to the device's small screen. The designer can specify that certain grid columns (e.g. an ID) or rows (e.g. a title bar) are shown on each split-screen for clarity.

In **Figure 7**, the screen layout design (1) shown is a complex one which is to provide an interface to display a list of jobs to be performed by a staff member. The rows and columns for the screen design (2) can be changed (3) in number and size (width for columns and height for rows). Each cell may be further refined into a number of differently-sized sub-rows and columns if required, providing GridBag-style layout control. Items can be placed in cells by drag-and-drop from the tool bar at left into the screen layout-based design (4). The designer can change the number of rows and columns for the design, their sizes, and the number of rows and columns for each cell as they require. Screen items in each row may span multiple columns (as shown in this example). Properties of screen elements may be modified in a dialogue box or in the tree-structured design view. Changes made to a design in the screen layout view are immediately

reflected in the tree structure and textual views and vice-versa. Designers use the screen layout view to specify interactively and view the two-dimensional organization of their adaptive interface designs.

As the user interfaces designed in our tool are adaptive, designers can never be sure just what an implemented interface will look and feel like for particular combinations of display device, user and user task, even using the screen layout view. In order to help them visualise their designs in a concrete fashion, our tool provides two views of a running interface design – one using a browser (IE6) and one a PDA/mobile phone simulator (the Nokia Mobile Toolkit). When the designer selects one of these views, the interface design is translated into an XML encoding and saved, and then an AUIT implementation of the interface design generated from it. A Tomcat JSP server is run and then the IE6 browser or Nokia Toolkit is then used to display the interface embedded in a tabbed panel in our tool (using ActiveX controls). Examples of these running interfaces are shown in **Figure 8**. The designer can interact with the running interfaces, and, depending on how much of the design and implementation they have completed, get a feel for the interface's suitability.

In **Figure 8** three interfaces are being run. A customer login screen (1) shows the appearance of this interface after specifying a particular user and task and this is rendered using an IE6 browser, set for full (800x600) screen resolution. In (2) a job list interface is being shown in IE6 with the display size set for 100x150 (a PDA-sized display device) using XHTML. The user is set to a job manager doing a job assignment task, allowing the manager to assign new jobs and modify job assignments.

The GridBag layout has been implemented by the generated AUIT JSP implementation of the interface using embedded HTML tables. This same interface is shown in (3) displayed using a spawned phone simulator which displays a WML-encoded version of the interface, using the same generated AUIT JSP page to generate the WML to display to the user and consume user inputs POSTed to the JSP page. In this example the user is a job performer who is browsing job details, resulting in no "Assign New Job" ability and no ability to modify job assignments.

We have implemented the code generation component of our tool using XSLT transformation scripts. When the designer wants to view an interface in one of the target implementation views (browser or PDA/mobile phone simulator) the shared design data model is translated into an XML encoding (currently the same used in the textual view editor) and saved to a XML file. We have currently implemented a full code generator to translate this XML encoding into an AUIT-augmented JSP page implementation, and partial proof-of-concept code generators to translate the XML-encoded design into standard JSP and servlet implementations. We chose to use XSLT scripts to carry out our code generation as these come with useful infrastructure in terms of XML parsing and output document generation support. They are also able to be readily edited, tested and extended without the need to change any code in our design tool implementation. New generation scripts can be seamlessly added that consume the same XML encoded design but generate vastly different target adaptive interface implementation.

Figure 9 shows an example of the translation of an XML encoded design (1) into an AUIT JSP page implementation (3) via an XSLT transformation script (2). This translation is pretty straightforward, as the design model data structure and its XML encoding were based on AUIT's custom tag library components and their parameters (Grundy and Zou, 2002). The translation of the XML design encoding into e.g. device-specific hard-coded Java Servlets is more complex, with the XSLT scripts basically embedding parts of the AUIT custom tag library component code into the generated Servlet .java file.

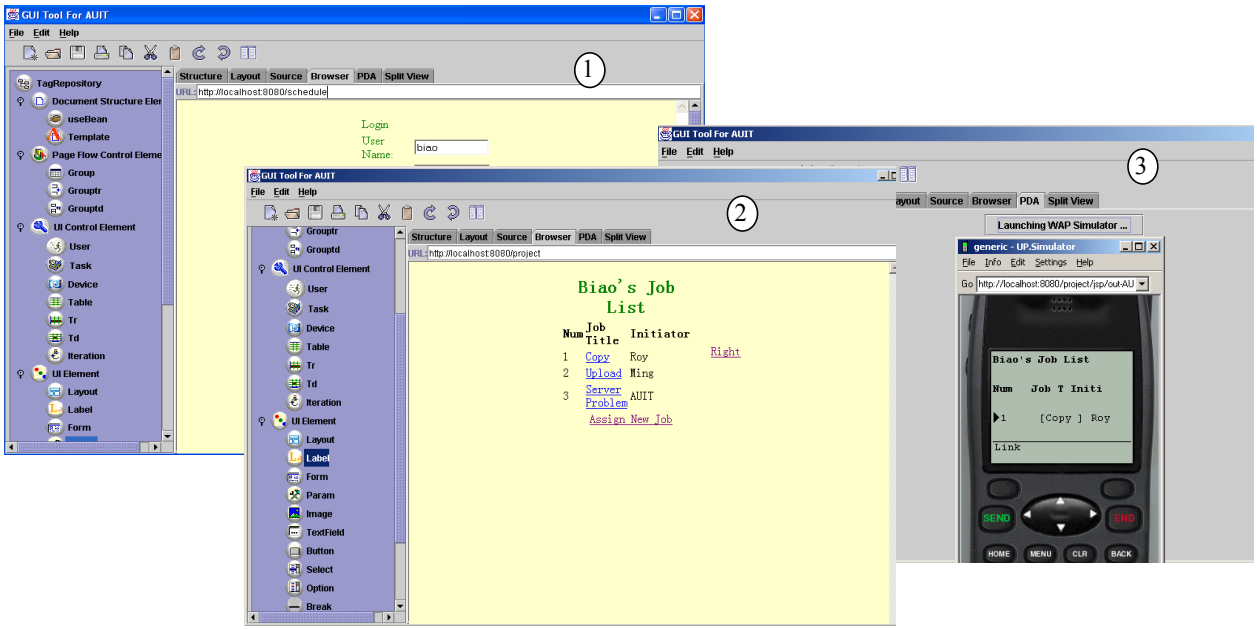


Figure 8. Examples of running generated AUIT interfaces.

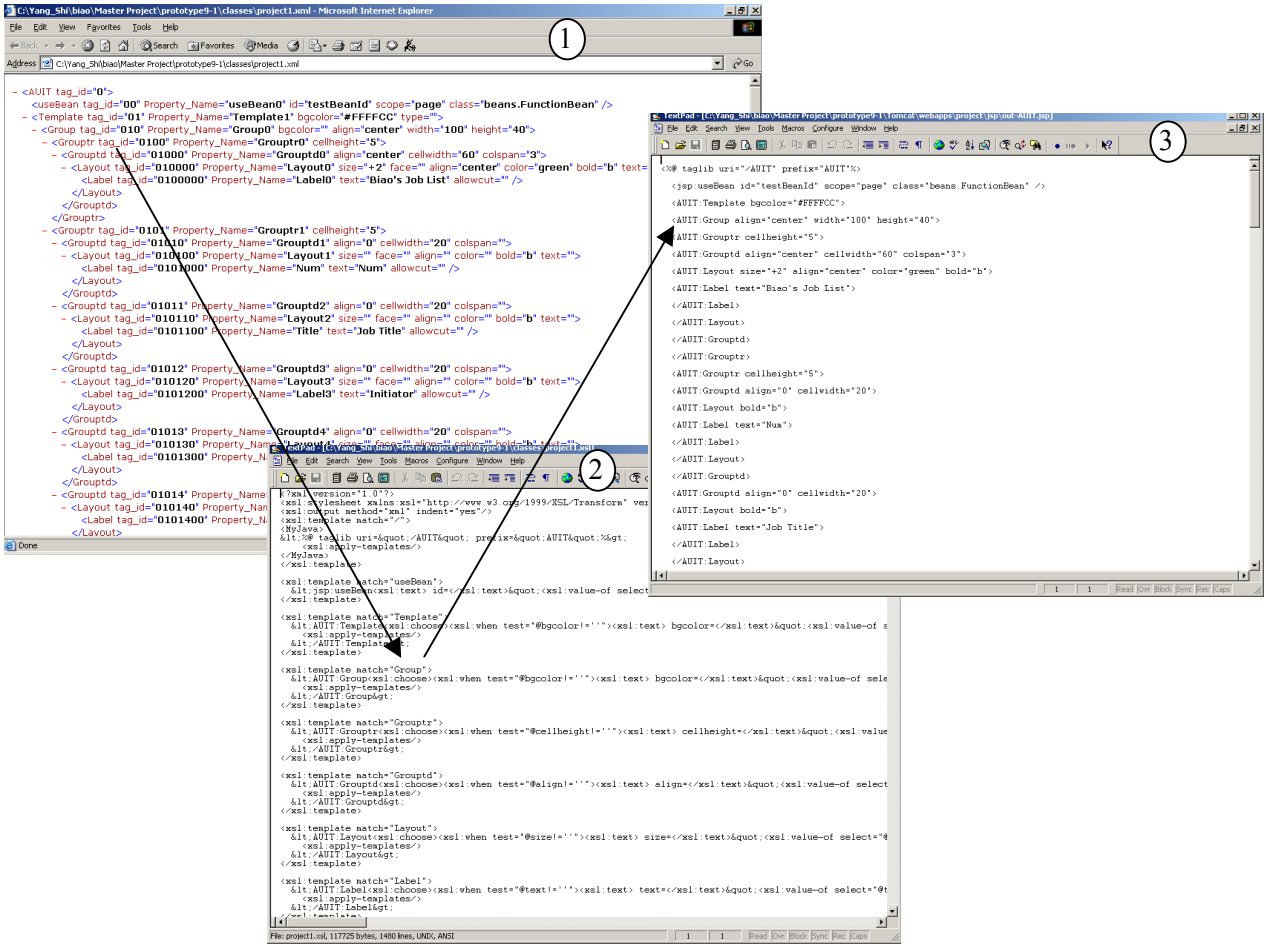


Figure 9. Examples of XML-encoded interface design, XSLT script and generated AUIT JSP page.

6 Discussion

We initially developed our design tool to allow developers of adaptive user interfaces to more easily design AUIT-based implementations. To this end we provided the tree-based logical structure view that closely corresponds to AUIT's XML-style hierarchy of screen elements, and provided a screen layout view using tiled grids to encapsulate screen elements and to support specifying and viewing their relative layout characteristics via direct manipulation and two-dimension rendering respectively. An XML encoded textual view closely corresponding to AUIT JSP page custom tags provides an alternative mechanism for expert users to view and edit detailed screen element properties. We have used our design tool to re-implement adaptive user interfaces for an on-line car site, an on-line video store and a collaborative job management system, all having been previously implemented with AUIT with no tool support. We are also planning to re-implement the interfaces for an on-line travel system and collaborative work support components using our new development tool.

Currently our design tool allows all the facilities provided by AUIT to be used in the tree-structure logical design view and the screen layout view. The tree structure view was developed initially as users of AUIT in previous studies indicated this would be a useful way of designing AUIT-based adaptive interfaces, due to the hierarchical way AUIT custom tags implementing adaptive interface elements are used. We have found that it does provide a useful visualisation of designs and provides an easier way of combining low-level screen elements to form an interface specification. It is very hard, however, to gain an idea of the likely appearance of anything but very simple screen designs with this view. The screen layout view was developed to overcome this by providing a tiled grid editor using AUIT's grid approach to interface layout specification. This view works well in allowing direct manipulation specification of layout, proving much easier than in the tree or textual views. However, we have also found that the current prototype of the screen layout view doesn't sufficiently convey the embedded griding structure of the design and needs to be enhanced to explicitly show designers the grid lines. We have found the textual view useful for detailed or multiple, different element property specification but little else. Both the tree and screen layout views prevent syntax errors in designs which is very useful given their complexity. The views of running implementations of a adaptive user interface design have proved very valuable in providing fast round-trip feedback to designer within the environment.

Improvements to our design tool we are planning include the ability to more easily associate parts of an interface with particular user roles and user tasks. We plan to use colour to distinguish interface components and groups applicable to user/task combinations. Many elements in our design tool provide "adornment" of fundamental

interface elements by enclosing the fundamental elements inside these adornment tags. For example the layout element is used to specify additional properties for elements it encloses, like text-fields, labels and buttons. This approach works well for implementing this adornment of screen elements using AUIT custom tag library classes but has the disadvantage of creating quite quite deep, complex hierarchies (as illustrated in some of the example design views in previous sections). This approach to specifying complex screen element properties was derived from AUIT's hierarchical tags and a better approach for interface design is probably to add these as properties to screen elements in their property dialogue box or to allow direct manipulation of these properties using tool bar and menu options, as done in word processors and some other GUI design tools. If generating AUIT implementations then the various AUIT adornment tags need only be created at code generation time. This would also allow quite different target implementations for adaptive interfaces to be more easily generated e.g. code for a Servlet which is organised quite differently to AUIT tags. We are currently designing a usability evaluation of our design tool to gather more information about its facilities and their appropriateness for adaptive interface design. Our intention is to survey several experienced developers of web-based and mobile user interfaces as we did when evaluating AUIT's effectiveness in previous work [0]. Many of our original set of developers who evaluated AUIT for us indicated the need for a development environment and we used their feedback when designing many of the facilities in the prototype environment described in this paper.

7 Summary

We have developed an implementation technology for building thin-client user interfaces that adapt to multiple display devices, user roles and user tasks. To aid developers using this technology we have developed a prototype design environment providing tree-structured, tiled screen layout and textual views of interface designs. Designers may edit any view with all others kept consistent – typically designers specify screen layout via direct manipulation with the screen layout view, detailed element properties with the tree-structured view and embedded Java code scriptlets for interface behvaiour with the textual view. Interface descriptions are encoded in XML and XSLT transformation scripts used to generate AUIT (or other) interface implementations. Developers may view and interact with running adaptive interface implementations via embedded views. We are currently designing a usability experiment to gather feedback on our design tool's performance and to guide further enhancement of it.

8 Acknowledgements

Wendy Zou's efforts in developing the AUIT adaptive user interface toolkit during her MSc study are gratefully acknowledged. We also thank the anonymous reviewers for their helpful comments.

9 References

AMOROSO, D.L. AND BRANCHEAU, J. (2001): Moving the Organization to Convergent Technologies: e-Business and Wireless, *Proc. 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. (2000): A. Web modelling Language (WebML): a modelling language for designing web sites, *Computer Networks* **33** (1-6).

DEWAN, P. AND SHARMA, A. (1999): An experiment in inter-operating, heterogeneous collaborative systems, *Proc. 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, *Proc. 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 Publishers.

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* **14** (3), Elsevier, 175-194.

GRUNDY, J.C. AND ZOU, W. (2002): An architecture for building multi-device thin-client web user interfaces, *Proc. 14th Conference on Advanced Information Systems Engineering*, Toronto, Canada, May 29-31 2002, Lecture Notes in Computer Science.

HAN, R., PERRET, V., AND NAGHSHINEH, M. (2000): WebSplitter: A unified XML framework for multi-device collaborative web browsing, *Proc. of Computer-Supported Cooperative Work 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 Proc. International Conference on Software Engineering, May 2001, IEEE CS Press, pp. 475-484.

MORCH, A. (1998): Tailoring tools for system development, *Journal of End User Computing* **10** (2), pp. 22-29.

ORACLE CORP. (1999): 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.

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, *Proc. 2001 Workshop on Multiple User Interfaces over the Internet*.

ZARIKAS, V., PAPATZANIS, G., AND STEPHANIDIS, C. (2001): An architecture for a self-adapting information system for tourists, *Proc. 2001 Workshop on Multiple User Interfaces over the Internet*.