An Architecture for Building Multi-device Thin-Client Web User Interfaces

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 information systems. Developers specify web-based interfaces using a high-level mark-up language based on the logical structure of the user interface. 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 much more easily construct and maintain adaptable web-based user interfaces than other current approaches.

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 [11, 8]. Interfaces need to be provided for conventional web browsers as well as wireless PDAs, mobile phones and pagers [8, 7, 12]. Adapting to different user and user tasks is required [4, 6, 11], such as hiding "Update" and "Delete" buttons if the user is a customer or a staff member doing an information retrieval task.

Building such interfaces using current technologies is difficult, time-consuming and systems are hard-to-maintain. Various approaches have been developed to support forms of user interface adaptation. Proxies such as Web Clipping™ and Portal-to-go services automatically convert e.g. HTML content to WML content for wireless devices [8, 7, 11, 9]. 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 [8, 11]. The degree of adaptation supported is generally limited, and each interface type requires complex scripting. Intelligent, adaptive and component-based user interfaces often support user and task adaptation [6]. 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 [11, 7] use generic, device-

A. Banks Pidduck et al. (Eds.): CAISE 2002, LNCS 2348, pp. 728-732, 2002.

independent user interface descriptions, but most do not typically support user and task adaptation and many are application-specific.

2 Our Approach

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 [1, 8] but without developing versions of every user interface for every possible device, user and user task comnination possible. To support this our 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"). 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 systems adopt the four-tier software architecture illustrated in Fig. 1.

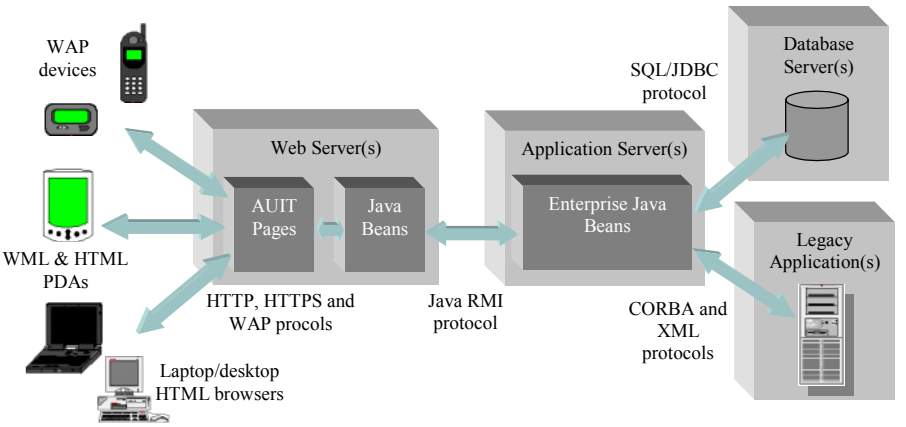


Fig. 1. Our 4-tier web-based information system software architecture

The AUIT pages are implemented by Java Server Pages (JSPs) containing a special mark-up language independent of device-specific rendering languages like HTML and WML but that contain descriptions of screen elements, layout and user/task relevance. Developers implement their thin-client web screens using this 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. At run-time, accessed AUIT pages determine the requesting display device type, current user and current user task. A display mark-up language suitable for the device is generated, taking into account the user and user task. Dynamic page content is via JavaBeans, connected to Enterprise JavaBeans accessing databases and possibly legacy systems.

3 An Example of AUIT in Use

We illustrate the use of our AUIT system used to build some adaptable, web-based user interfaces for a collaborative job maintenance system. Fig. 2 shows examples of a 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 screens. The user accesses these additional details by using the hypertext links added to the sides of the screen. This interface splitting is done automatically by our AUIT tag implementation and uses the logical structure of screens to ensure a sensible re-organisation. 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 and scripting the PDA and web browser can, so buttons, links, colour and client-side scripts are not used. The user accesses other pages via a text-based menu listing. If a customer job manager accesses job lists and details, they can change the details whereas another staff member cannot (Update buttons/menus are hidden). Similarly, if a job manager accesses job details when "viewing" a job (versus updating it), these are hidden.

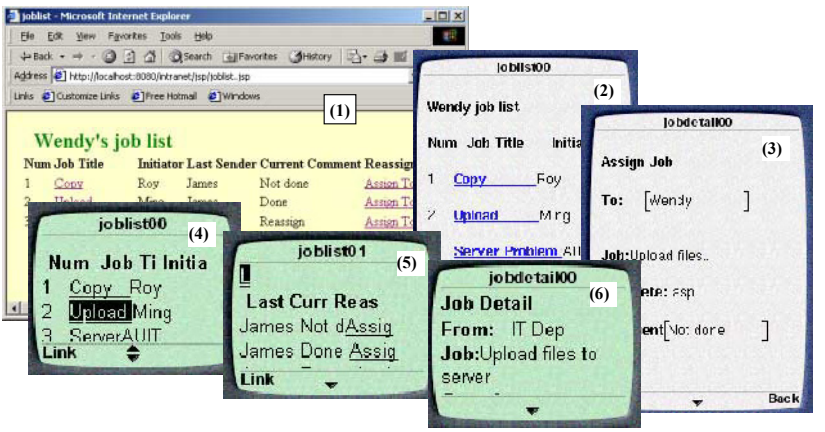
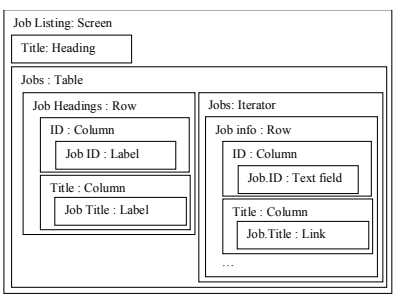


Fig. 2. (a) Examples of job listing screen running on multiple devices




```
<%@ taglib uri="/auit" prefix="auit" %> // page directive to access AUIT tags
<jsp:useBean id='job_manager' class='jobs.JobManager />
<auit:screen name="job list"> // sets user/task/device information…
  <auit:heading level=2 value='<%= AUITUser.getName() %>'s Job List' />
  <auit:table width=60 border=0>
    <auit:row><auit:column><auit:label width=6 value='Num' />…
    <% 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>
        …
```

Fig. 3. (a) Logical structure of and (b) example AUIT description parts of the job listing screen

Fig. 3 (a) shows the logical structure of the job listing screen using our AUIT custom tags. 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. Fig. 3 (b) shows part of the AUIT Java Server Page that specifies this interface. These AUIT tags generate HTML or WML mark-up as output, adapting the layout, use of graphics and highlighting, available buttons and menu items, data fields shown and so on to the requesting device, user and user task information supplied when the page is accessed. Use of a logical structure for web-based interfaces is essential to this approach, as in related approaches [3, 11].

We have carried out an empirical evaluation of both the AUIT tag library by experienced web developers and the AUIT-generated user interfaces, comparing them to interfaces from custom-written JSPs. Results have been very encouraging with developers able to build quite sophisticated web-based information systems whose interfaces suitably adapt to a variety of display devices, users and user tasks. We are continuing to refine and extend the custom tags and their properties to allow more complex layout, highlighting and client-side scripting to be adapted.

References

1. Amoroso, D.L. and Brancheau, J. 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.
2. Bonifati, A., Ceri, S., Fraternali, P., Maurino, A. 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.
3. Ceri, S., Fraternali, P., Bongio, A. Web modelling Language (WebML): a modelling language for designing web sites, Computer Networks **33** (1-6), 2000.
4. Eisenstein, J. and Puerta, A. Adaptation in automated user-interface design, Proc. 2000 Conference on Intelligent User Interfaces, New Orleans, 9-12 January 2000, ACM Press, pp. 74-81.
5. Fox, A., Gribble, S. Chawathe, Y., and Brewer, E. Adapting to Network and Client Variation Using Infrastructural Proxies: lessons and perspectives, IEEE Personal Communications **5** (4), (1998), 10-19.
6. Grundy, J.C. and Hosking, J.G. Developing Adaptable User Interfaces for Component-based Systems, to appear in Interacting with Computers, Elsevier (2001).
7. Han, R., Perret, V., and Naghshineh, M. WebSplitter: A unified XML framework for multi-device collaborative web browsing, Proc. CSCW 2000, Philadelphia, Dec 2-6 2000.
8. Marsic, I. Adaptive Collaboration for Wired and Wireless Platforms, IEEE Internet Computing (July/August 2001), 26-35.
9. Palm Corp. Web Clipping services, www.palm.com (2001).
10. Rossel M. Adaptive support: the Intelligent Tour Guide. Proc. 1999 International Conference on Intelligent User Interfaces. ACM Press.

11. Van der Donckt, J., Limbourg, Q., Florins, M., Oger, F., and Macq, B. Synchronised, model-based design of multiple user interfaces, Proc. 2001 Workshop on Multiple User Interfaces over the Internet, HCI-IHM'2001, 10-14 September 2001 Lille, France.
12. Zarikas, V., Papatzanis, G., and Stephanidis, C. An architecture for a self-adapting information system for tourists, Proc. 2001 Workshop on Multiple User Interfaces over the Internet, HCI-IHM'2001, 10-14 September 2001 Lille, France.