Experiences developing a thin-client, multi-device travel planning application

John Grundy[1, 2] and Weiguo Jin[2]
[1]Department of Electrical and Electronic Engineering and [2]Department of Computer Science
University of Auckland, Private Bag 92019, Auckland, New Zealand
+64-9-3737-599 ext 8761
john-g@cs.auckland.ac.nz

ABSTRACT

Many applications now require access from diverse human-computer interaction devices, such as desktop computers, web browsers, PDAs, mobile phones, pagers and so on. We describe our experiences developing a multi-device travel planning application built from reusable components, many of these developed from several different previous projects. We focus on key user interface design and component adaptation and integration issues as encountered in this problem domain. We report on the results of a useability evaluation of our prototype and our current research directions addressing HCI and interface development problems we encountered.

Keywords

Multi-device user interfaces, web-based applications, groupware, adaptive user interfaces

INTRODUCTION

The advance of Internet technology stimulates and accelerates the development of many new applications. Computer Supported Collaborated Work, CSCW, is one such application that leads to multiple-user collaboration across time and space constraints. Using computer networks as the computing infrastructure, groupware is a collection of software components that inter-operate together to support a group of users to achieving a shared set of tasks. In this project, we explored some interesting issues in this area by prototyping a thin-client, component-based travel planning groupware system. This allows a travel agent and multiple customers to collaboratively plan travel itineraries using a range of heterogeneous web-based interfaces, including web browsers, PDAs and mobile phones.

Group work occurs in all work places. It can take place either within the same location or distributed locations synchronously or asynchronously. A large number of groupware applications have been developed, examples including chat, email, ICQ, video and audio conferencing, collaborative document editors and shared calendars [1, 4, 14]. Most of these systems are desktop interface-oriented and generally have a limited range of functionality. It is very challenging to engineer groupware and to sufficiently integrate it with other applications that users require [5, 10]. One solution is the use of component-based technologies that offer improved mechanisms for integrating reusable parts of systems.

We built our travel planning system based on the following scenario: multiple users are working together to plan a travel itinerary - create and modify itineraries, search for flights, hotels and rental cars, make and modify bookings and so on. They are supported by groupware facilities, which include communication, coordination and collaboration facilities. Examples of these facilities would ideally include synchronous video/audio, semi-synchronous chat, asynchronous email/messaging, document annotation, notification events, group awareness, and version control. When developing this system we did not build it from scratch. Instead we reused a set of reusable groupware components (supporting as chat, email, notification and annotation facilities) [5], a web-based travel planner application, and an adaptable, multi-device interface implementation technology [6]. All of these were developed for previous, unrelated projects. We had a number of successes and difficulties realising the collaborative travel planning application using this approach, from both HCI and user interface implementation perspectives.

MOTIVATION

In previous research projects we developed a number of applications:

- A collaborative travel planner [7], including a web-based version of this [5]. This had hard-coded groupware capabilities as well as support for managing travel itineraries (creating, searching, modifying, booking etc).
- A set of multi-device groupware components, including chat, email, note annotation, notification and to-do list [7, 5]. These were designed to be reused in any thin-client application that required such collaborative work support.
- A technology allowing developers to specify "adaptive" user interfaces, where one interface

specification can be adapted at run-time to different device, user and task needs i.e. the interface layout and composition changes depending on the device requesting it, the user using that device, and the user's current task [6].

We wanted to investigate combining these three research threads to produce a collaborative travel planner that would run on multiple devices, leverage our reusable groupware components, and make use of our adaptable user interface building technology to avoid multiple implementations of the same interface for different devices. Adaptive user interfaces ideally provide a way for developers to specify a complex interface once and have the implementation of this interface "adapted" to suit particular run-time display device, user and user task characteristics. Ultimately there is a trade-off between supporting easier interface specification and implementation and the usability of the resultant interfaces: a hard-coded, custom interface will always at least potentially be "better" than an adapted one from a single specification [15, 7].The aim of this research is to see to what degree these adapted groupware and application interfaces are deficient in terms of usability and improve their implementation technology to address these issues.

RELATED WORK

Many examples of groupware have been developed. Some key examples include messaging systems (e.g. email, ICQ, IRC), collaborative editing tools (e.g. Grove, DUPLEX, CocoDoC) [4], meeting support systems (e.g. MS Netmeeting™, TeamWave) [12], and workflow and work co-ordination systems. To date most groupware is thick-client (desktop) and custom-built for an application. However, in recent times the use of software components to build groupware [14, 10] and the use of "new" interaction devices and technologies has become popular in groupware research and applications. Examples of the later include the use of virtual reality interfaces (e.g. CHIME) [3], web-based user interfaces (e.g. MILOS, OzWeb and BSCW) [10, 1], and mobile devices [9]. Integrating such groupware with desktop applications is possible though limited, but integrating them with other VE, web or mobile applications more promising. A number of challenges present, particularly with small-screen mobile device groupware and applications and ideally all thin-client interfaces should provide consistent interfaces.

Building adaptive interfaces that can be displayed on multiple devices using current technologies is difficult and such 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 [2, 11]. 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 [2, 13]. 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 [7]. However 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 [15] use generic, device-independent user interface descriptions, but most do not typically support user and task adaptation and many are application-specific.

ARCHITECTURE

Our travel planning application and thin-client groupware use the architecture illustrated in Figure 1. The travel planner was built with a set of Java Server Pages providing web browser (HTML) user interfaces to register, login, create itineraries, search for flights, hotels etc and make and confirm bookings. A four-tier architecture was used with a set of business logic-implementing application server objects accessed via CORBA. The groupware components used a similar architecture but provided several alternative web server interface implementations, allowing for web browser, PDA and WAP client devices.

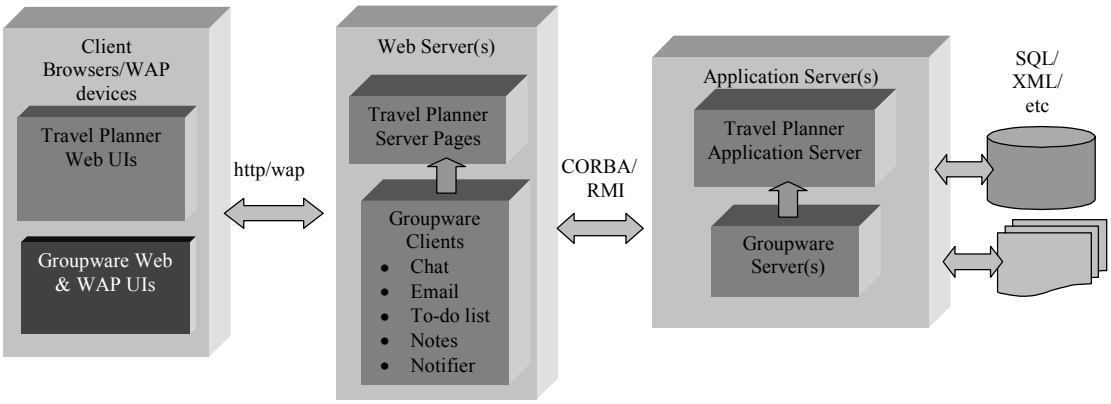

Figure 1. Architecture of our thin-client groupware.

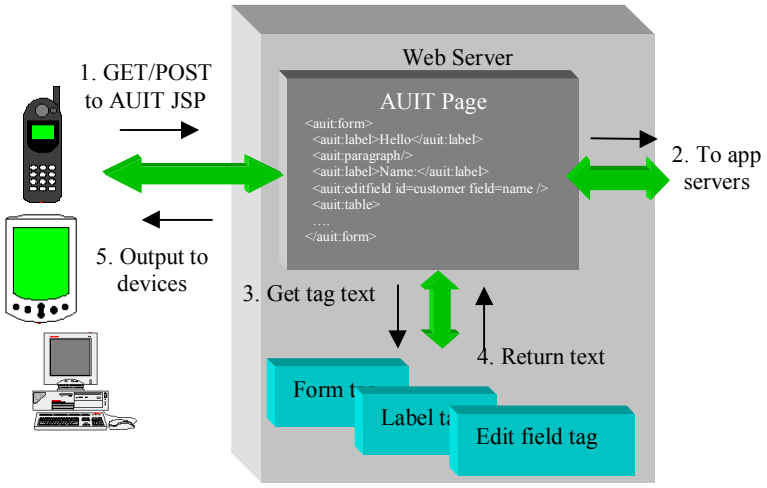

Figure 2. AUIT Page Processing.

The groupware components provided some simple access points to the travel planner application via frames. These allowed the users to add notes to travel planner interfaces and to view and send messages while working with travel planner interfaces.

The travel planner was not modified to support this interaction and as it was not originally designed to work with the groupware components, only limited user interface and server integration existed between them [5]. We found this to be a problem in terms of the level of user interface consistency that could be achieved between the applications and in terms of travel planner application events that could be subscribed to and acted upon in the groupware components. We wanted to provide a more seamless integration between both web user interfaces and application servers of both applications, while retaining their implementations' independence.

We have recently developed a new technology allowing developers to specify web-based user interface implementations that will automatically adapt to different display devices, users and user tasks, called AUIT [6]. This provides a set of custom tag libraries for Java Server Pages that allow developers to specify logical screen structure, composition and layout. At run-time these tags are interpreted and produce output for a Web or WAP browser tailored to the display device characteristics and the particular user using the device and the user's current work task

We wanted to use AUIT to re-implement the travel planner and groupware user interfaces, allowing the travel planner to be seamlessly accessed from different devices and support a degree of user and task adaptation. Similarly, we wanted to provide groupware user interfaces that would adapt to different devices without the need for a different interface implementation for every possible display device. The AUIT architecture is illustrated in Figure 2. User interface implementations are implemented using a custom Java Server Page tag library and these tags use device, user and task information to construct appropriate page mark-up output for page requests. Tags include page layout control (groups, tables, rows and columns), data input/output (text fields, pop-up lists, radio and checkboxes), navigation and control (buttons and links), and page embellishment (labels, lines, borders, images). The logical specification of a page may be at run-time split into multiple physical mark-up pages for small-screen devices, with navigation links between the pages generated [6].

INTERFACE DESIGN AND IMPLEMENTATION

Figure 3 illustrates some of our prototype travel planner interfaces, displayed in a web browser. These allow a user to create and modify travel itineraries, search for hotel rooms, rental cars, plane flights etc, and to view and print a confirmed itinerary. The interfaces use a fairly basic page layout and user interface elements. In (1), user John is viewing a set of travel itinerary items. In (2), he is entering search criteria to locate a required plane flight. In (3) he is selecting from a list of possible flights. These interfaces allow access to various groupware facilities provided by our reusable groupware components. In addition, the groupware components can subscribe to various travel planner application events and act on these e.g. perform various tasks based on event notifications received from the travel planner.

A number of different groupware components are reused in this travel planner. Some of the interfaces for these functions are illustrated in Figure 4. In screen (1) user Mark is reading a note annotation made by user John against a travel plan item. The note information is stored and maintained within the groupware Annotation component.

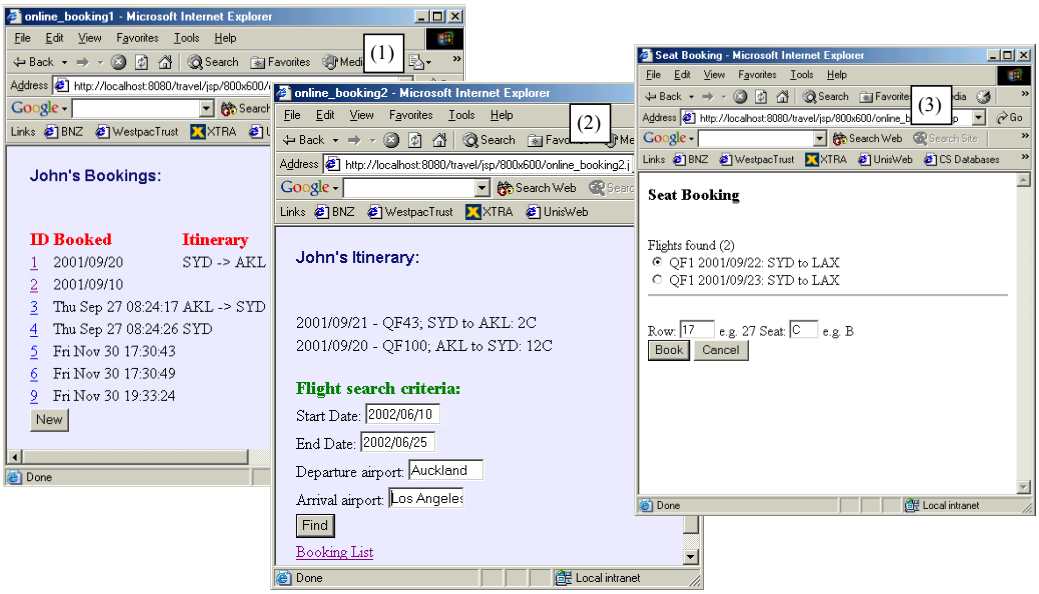

Figure 3. Examples of web-based travel planner user interfaces.

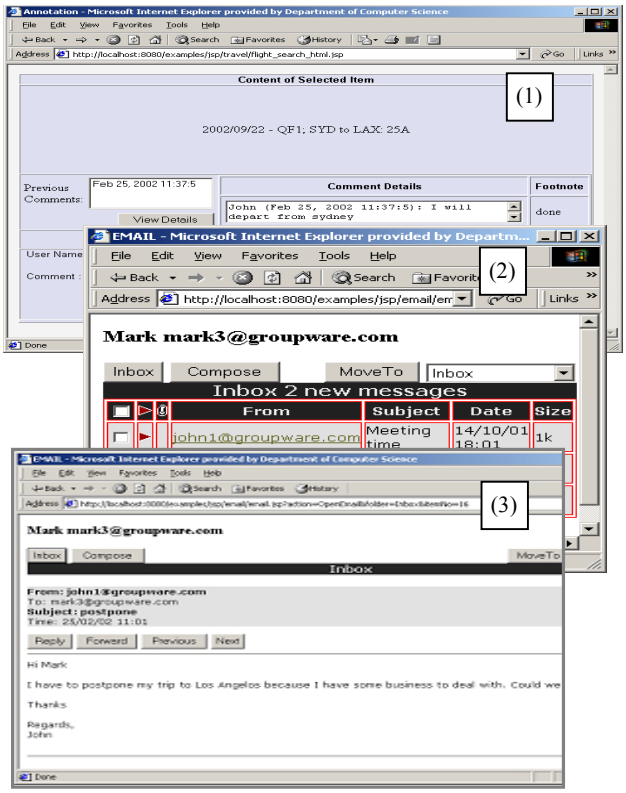

Figure 4. Examples of web-based groupware interfaces.

The travel planner web server and application server provide interfaces that allow the groupware components to determine which user, itinerary and item are to be noted. Access to the groupware facilities in the travel planner is via reusable JavaScript-implemented menus and by multiple frames grouping some groupware interfaces (such as notes, messaging and awareness).

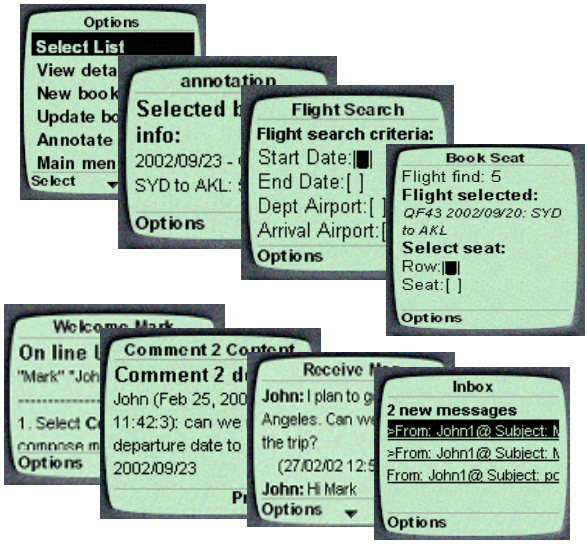

Figure 5. Examples of WAP interfaces.

In screen (2), Mark is viewing a list of received email messages, some sent by John and some by a groupware notification agent monitoring travel planner application activity. The users can configure the notification agent with their own desired events to be listened to and the way to have these acted upon. Screen (3) shows Mark reading an email message. Messages and notification events can be annotated with notes just like travel planner application information.

In addition to providing conventional web-based user interfaces, our prototype provides equivalent WAP implementations via the AUIT page implementation

technology. Some of these travel planner user interfaces are shown at the top of Figure 5, accessed via a mobile phone WAP client. The content of many of the different interfaces has been split across multiple "cards" allowing management of complex interface content. In these examples, user John is accessing his itinerary information on a mobile device, allowing him to both maintain this information and also access it before and during his trip. Some examples of our groupware component user interfaces in use by Mark via a WAP display device are shown at the bottom of Figure 5.

Figure 6 shows an example of part of an AUIT-implemented travel planner interface, the booking list screen. The mark-up language used is generic and at run-time is converted into HTML or WML for a display device. An algorithm splits too-large screens into parts for display on small devices.

```
<AUIT:template bgcolor="#EEEEFF">
<AUIT:group width="800" height="600" >
 <AUIT:form method="post" action="online_booking2.jsp" name="update">
<AUIT:grouptr  cellheight="20" >
  <AUIT:grouptd  cellwidth="150" colspan="3">
   <AUIT:layout  face="Arial, Helvetica, sans-serif" size="3" color="#101077" bold="b">
      <AUIT:label text="John's Bookings:" allowcut="true"></AUIT:label></AUIT:layout>
</AUIT:grouptd> </AUIT:grouptr>

 <AUIT:grouptr><AUIT:grouptd>
     <AUIT:break/>
    <AUIT:layout size="+1" color="red" bold="b"><AUIT:label text=
        "<%= booking_interface.getMessage() %>" /></AUIT:layout>
</AUIT:grouptd></AUIT:grouptr>

 <AUIT:grouptr>
  <AUIT:grouptd><AUIT:layout size="+1" color="red" bold="b">
      <AUIT:label text="ID"></AUIT:label></AUIT:layout></AUIT:grouptd>
  <AUIT:grouptd><AUIT:layout size="+1" color="red" bold="b">
      <AUIT:label text="Booked"></AUIT:label></AUIT:layout></AUIT:grouptd>
…

 <% Vector bookings = booking_interface.getBookings(customer_data); %>
 <AUIT:iterator collection="<%=bookings%>" >
 <% BookingData booking = (BookingData)bookings.get(index.intValue()); %>
 <AUIT:grouptr>
   <AUIT:grouptd cellwidth="30">
      <AUIT:link direct="online_booking2" param="true">
                <AUIT:label text='<%=""+booking.getID()%>'></AUIT:label>
                <AUIT:param name="ID" value='<%=""+booking.getID()%>'></AUIT:param>
       </AUIT:link>
   </AUIT:grouptd>
   <AUIT:grouptd cellwidth="80">
      <AUIT:label text='<%=""+booking.getDateBooked()%>' allowcut="false"></AUIT:label>
    </AUIT:grouptd>
    <AUIT:grouptd cellwidth="60">
                  <%
                   String itinerary = "";
                  …
                  %>
                 <AUIT:label text='<%= itinerary %>' allowcut="false"></AUIT:label>
 </AUIT:grouptd> </AUIT:grouptr>
 </AUIT:iterator>
…
```

Figure 6. Example of AUIT-implemented screen.

EXPERIENCES

We has previously carried out useability evaluations of our travel planner application, groupware components and two AUIT-implemented applications [5, 6, 7]. We now wanted to assess our new combined prototype's useability. We wanted to focus particularly on:

- determining if our integrated application provides a suitable set of interfaces and functionality for collaborative travel planning
- assessing the users' ability to navigate between screens and between travel planner and groupware functionality seamlessly
- assessing the effectiveness of our auto-adaptation to device characteristics

We carried out a usability evaluation of our original travel planner and groupware prototypes by surveying users, both experienced and novice when using the applications in a collaborative setting [5, 6]. We assessed our new prototype by developing criteria to rate the original and new prototype interfaces with respect to the above usability measures. The key advantages of our new approach to engineering the travel planner include the reuse of significant existing software to minimise development effort, the aim of reusing all groupware and adaptive user interface building technology, and the approach to providing application software with interfaces that allow other software to more readily interact with them. Further work is required to improve the reusability of the groupware and to help designers develop the most suitable application interfaces. In general, our evaluation indicated the interfaces provided were both effective and efficient for the tasks they were required to support.

The main problems we encountered in this research were concerned with the ability to adequately generalise the groupware component user interfaces and software interfaces to enable them to be adequately integrated with other application components. Similarly, we found we had to hard-code links to groupware screens and frames to contain parts of groupware interfaces in the travel planner application. The AUIT custom tag library proved to be useful for building interfaces that automatically supported multiple devices. However the range of layout control facilities and screen components proved less than adequate. The small screen interface implementations had almost always to be split at run-time, though access to multiple screens via additional option links worked reasonably well. Careful specification of the "priorities" and grouping of screen components is needed, however, to ensure sensible screen grouping results. We found our original groupware component screen implementations to provide better interaction and layout support than the AUIT-implemented versions, due to being tuned to the particular device characteristics. However, we feel some simple improvements to our AUIT tags will enable better quality interfaces to be produced that better approximate the device hand-coded ones. The prototype's performance can be rather slow, and we have found this to be due to some inefficiencies in our AUIT screen formatting code.

We are currently making a number of extensions to the AUIT custom tag library to provide better layout control, more developer control over screen splitting choices based on relative "importance" of screen elements, and to provide faster interactive performance via caching. We are also building a GUI design tool to allow much easier design, implementation and testing of AUIT-based thin client interfaces. We are redesigning the groupware components to have improved component integration. This especially focuses on allowing third party thin client applications to integrate groupware functionality and the groupware to

integrate the third party application interface components. In addition, the travel planner application we aim to further generalise and provide complete separation from the groupware components, allowing others' groupware support to be used instead. One approach we are experimenting with is to add further AUIT custom tags that allow the interface developer to incorporate screens from other applications within an AUIT-specified interface. This would allow e.g. the note annotation and messaging groupware interfaces to be incorporated within the travel planner without hard-coding this relationship in the travel planner or groupware interface implementations. We did not use the AUIT user and task adaptation support features in this work. However some interface functions and information display is user and task-dependent in the travel planner application. We could use these facilities to avoid complex if-then-else constructs in the user interface implementations and avoid having different implementations of the same basic user interface for different users.

SUMMARY

We have developed a prototype collaborative travel planner from three separate applications: a web-based travel planner; a set of hard-coded multi-device groupware components; and a Java Server Page custom tag library that allows developers to specify a user interface once that will at run-time adapt to producing device-specific display mark-up. This was moderately successful, with a range of travel planner and groupware interfaces being re-engineered and implemented to make use of this adaptive user interface specification technology. The travel planner and groupware components had minimal knowledge of each other, both in terms of their user interface implementations and their business logic component implementations. Evaluation of the prototype showed it provides a reasonably good set of interfaces to support collaborative travel planning using multiple devices. However, we found that both applications needed further generalisation to separate their interface dependencies. In addition, we could use our adaptive interface technology's support for user and task adaptation in these two sets of interfaces.

REFERENCES

1. Appelt W. WWW based collaboration with the BSCW system, *Proceedings of the 26th Conference on Current Trends in Theory and Practice of Informatics*, Springer-Verlag LNCS 1725, 1999, pp.66-78.
2. Chong, N.S.T., Sakauchi, M. e-CoBrowse: co-navigating the Web with chat-pointers and add-ins - problems and promises. *Parallel and Distributed Computing and Systems.* vol.2, 2000, IASTED/ACTA Press, pp.803-808.
3. Dossick, S.E. and Kaiser, G.E. CHIME: A Metadata-Based Distributed Software Development Environment. *Joint Seventh European Software Engineering Conference and Seventh ACM SIGSOFT International Symposium on the Foundations of Software Engineering*, September 1999, pp. 464-475
4. Ellis, C.A., Gibbs, S.J. and Rein, G. Groupware: some issues and experiences, *Communications of the ACM*, vol. 34, no. 1, January 1991, pp. 39-58.
5. Grundy, J.C., Wang, X., Hosking, J.G., Building Multi-device, Component-based, Thin-client Groupware: Issues and Experiences, *In Proceedings of the 3rd Australasian User Interface Conference*, Melbourne, Australia, 28-30 Jan 2002, CRPIT Press.
6. Grundy, J.C. and Zhou, W., An architecture for building multi-device thin-client web user interfaces, In *Proceedings of the 2002 International Conference on Advanced Information Systems Engineering*, Toronto, Canada, Lecture Notes in Computer Science, Springer.
7. Grundy, J.C. and Hosking, J.G., Developing Adaptable User Interfaces for Component-based Systems, *Interacting with Computers*, Elsevier, May 2002.
8. 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.
9. Hartmann S, Dirksen V. Optimization of internal business processes through integration of mobile commerce components. *Information Management & Consulting*, vol.16, no.2, May 2001, pp.16-19.
10. Maurer F, Dellen B, Bendeck F, Goldmann S, Holz H, Kotting B, Schaaf M. Merging project planning and Web enabled dynamic workflow technologies. *IEEE Internet Computing*, vol.4, no.3, May-June 2000, pp.65-74.
11. Palm Corp. *Web Clipping services*, www.palm.com (2001).
12. Roseman, M. and Greenberg, S. TeamRooms: network places for collaboration, *Proceedings of the ACM 1996 conference on Computer supported cooperative work*, 1996, Pages 325 - 333.
13. Rossel M. Adaptive support: the Intelligent Tour Guide. *Proc. 1999 International Conference on Intelligent User Interfaces*. ACM Press.
14. Shuckman, C., Kirchner, L., Schummer, J. and Haake, J.M. 1996. Designing object-oriented synchronous groupware with COAST, *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, ACM Press, November 1996, pp. 21-29.
15. 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*.