ACCESSING WEB APPLICATIONS WITH MULTIPLE CONTEXT-AWARE DEVICES

ELMAR BRAUN
GERHARD AUSTALLER
JUSSI KANGASHARJU
MAX MÜHLHÄUSER
Telecooperation Group
Department of Computer Science
Darmstadt University of Technology
Hochschulstrasse 10, 64289 Darmstadt, Germany
E-mail: {elmar, gerhard, jussi}@tk.informatik.tu-darmstadt.de
E-mail: max@informatik.tu-darmstadt.de

Current web applications are meant to be accessed from a single device, typically a desktop computer. Single authoring has been proposed as a solution for allowing web applications to be used from a variety of devices. However, single authoring does not provide sufficient flexibility to allow a user to roam seamlessly from one device to another. In this paper, we address this problem and present our architecture for allowing access to web applications from a set of federated devices. Our architecture is intended as a general-purpose solution and is able to select the appropriate devices, synchronize the different views, and transcode documents as needed. We also present the main features of the implementation of our architecture, including an infrared positioning system which allows us to determine which devices are available to the user.

1 Introduction

One problem of most current web applications and web sites is that they are not authored in a device independent manner. Using a device other than a desktop computer to render such markup will often result in a broken or unusable rendering, or not be possible at all. *Single authoring* (also referred to as device independent or platform independent authoring) has been proposed as a solution to this problem[1]. Web sites are specified in a single, device independent format, which can be used to generate adapted content for all available client devices automatically. The vision of device independence is that users can switch between devices at will, and find a usable and familiar representation of their web applications on every device.

Single authoring is an interesting way of matching the research challenges of *ubiquitous computing*. Ubiquitous computing envisions that computing and interaction facilities will be generously integrated in the environment, and available to be used casually by everyone[2]. Whenever a user needs the services of a computer, she simply picks up or walks up to any device that she finds convenient. She should not have to consider which device will render her applications best, as this conflicts with the requirement that use is casual and effortless.

In such environments, it becomes clear that single authoring alone is not sufficient. Single authoring provides a usable interface on any device, but migrating from device to device still requires the user to authenticate herself to the machine, and to browse to her personal web application from this unpersonalized device. If the user wishes to roam to a different device, she has to undergo the same ordeal there. If she had already filled out parts of a web form, she cannot roam to another device without having to repeat all input on the new device.

In this paper, we present our solution to extend the possibilities of single authoring to multiple, federated devices. Our approach to this problem consists of four steps. The first step is to use context awareness techniques to monitor which device a user is currently using. The second step is to store the state of a web application on a server while it is being filled out. When a user roams from one device to another, this is detected by the context sensors, and the currently open web application is migrated by the server from the old to the new device.

This enables us to dispose of the requirement that the user only browses from one device. This is the third step of our solution. A particular focus of our research considers scenarios where the user roams between a mobile device and a larger, fixed device. Consider the example of roaming from a PDA to a wall-mounted screen. Simple roaming would allow the user to move her running session from the PDA to the display, so that she gains the advantages of the larger screen but loses the ability to make input through her PDA. If the display features no convenient mode of input, then the quality of the presentation of the web application actually becomes significantly lower by roaming. We therefore allow coordinated access to a web application from multiple, federated devices, which work together to display a single web application. So in the above example, the user would still be able to perform input on the PDA, while receiving output and feedback from the larger, better display. The benefit of such an approach becomes even more obvious when you replace the PDA with a speech-based device. Interaction becomes multimodal – it takes place through speech and a display at the same time.

A naive approach to such device-spanning presentations of web applications would be to display the full application on each device concurrently. However, this has the major drawback that some devices may not support all the components in the interface. For example, a large wall-display might not have any input capabilities, and a speech-based device is unable to render images in a meaningful manner. Instead of naively displaying all of the interface on all devices, we try to generate a user interface for each device which explicitly takes into account that there are other devices, which might be better suited to render certain parts of the web application. Therefore, the fourth step in our approach is an explicit support for such splitting of user interfaces.

2 Architecture

Figure 1 illustrates our architecture. Sensors detect which devices (both personal and publicly available) are able to interact with the user. This information is processed and stored by a context server implementing our context stack model. An application server (the *dialog manager*) uses this information to decide through which device or devices it will interact with the user. Then it transcodes user interface markup to the device specific markup required by the various clients. It also maintains a consistent state among the different views which the different clients present.

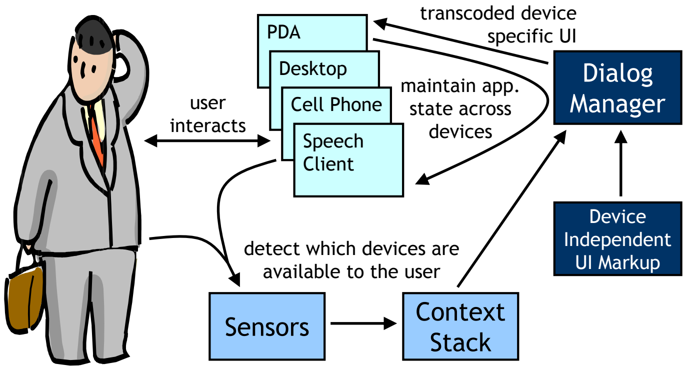
PDA
Desktop
Cell Phone
Speech Client
transcoded device specific UI
maintain app. state across devices
Dialog Manager
user interacts
Device Independent UI Markup
detect which devices are available to the user
Sensors
Context Stack

Figure 1. Architecture sketch.

2.1 Multi-Device Browsing

In the introduction we have proposed to use multiple devices concurrently to render a single web application. In a way, these different devices form a "virtual browser". A traditional browser displays all content on one computer and even in the case of multimodal browsers, such as the multimodal browser developed by IBM and Opera[a], all channels are still rendered by a single program on a single device. When we move away from the desktop and its single, full-featured device, to an ad hoc federation of devices that cooperatively renders the various channels, some part of the browser has to move to the network as well, in order to orchestrate all these devices.

We call this server the dialog manager. Its tasks are to coordinate input and output occurring on any given channel with all other devices. In software engineering terms, the dialog manager can be understood as the model in a model-view-controller pattern, whose multiple views and controllers run on

[a]`http://www-306.ibm.com/pvc/multimodal/`

several distributed devices. We explain details of our implementation of this concept in Section 3.

The other task of the dialog manager is to listen to context events that tell it which devices are currently available to the user. Whenever this set of available devices changes, the dialog manager needs to generate a new set of sub-documents, adjusted to the new situation, from the document that it is supposed to render. This is because some devices in the set may not be able to render the full document. Even if they were capable of it, the result could be more confusing than helpful. For example, a PDA, when used in conjunction with a large screen display, would not render elements that take up much space. Instead it would primarily render the most important input elements, which will then be available on a device that is easy to reach as it is already in the hand of the user.

Segmenting an HTML document in such a manner is difficult, as it requires information about parts of the document (e.g., their order of importance) which are not usually provided by HTML. Therefore we have decided to make a small number of extensions to XHTML and XForms, which allow single authoring a document that can be rendered on an arbitrary set of devices. The dialog manager transcodes this source document to a set of concrete documents whenever its set of available devices changes. In other words, our "virtual browser" is the "virtual target device" for which the transcoder generates output. More information about our single authoring language can be found in earlier publications[3,4].

2.2 Context

The inclusion of information about the user and her context is a requirement for providing the users with seamless and effortless interaction. Context information enables us to determine which devices are currently easily usable by the user and therefore are suitable candidates for interaction.

Our context architecture provides context aware applications with a context stack[5], which covers the delivery of context information from the context sensors to the applications and provides possibilities for filtering at intermediate stages. We present an overview of the context stack, and how it can deliver the information we need to make decisions about how to display our applications, below.

Figure 2 presents our context stack. It is divided into five context layers and the application layer. Their roles are described below.

- The **Context Sensor** layers contains all the context sources which are of interest to us. The task of this layer is to feed context information to the rest of the stack. It is responsible for expressing the context information in a suitable format and also adds metadata (e.g., a timestamp) to the information. This metadata is passed up along the stack so that the higher layers can decide whether that piece of information is still relevant.

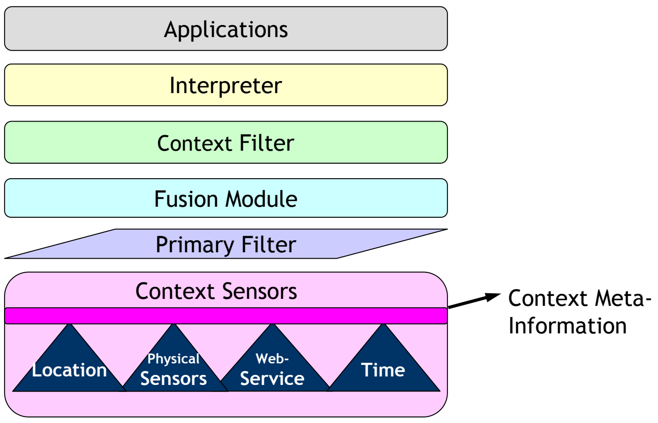

Figure 2. The Context Stack.

- The **Primary Filter** protects the stack from being flooded by information. When context information is continuously delivered from a source (e.g. a temperature reading), the application may be interested in only a few values every now and then.

- The **Fusion Module** merges the context information delivered through the Primary Filter. This can either make the context information easier to access (for the higher layers) or get a higher-level representation of the raw context information. Some abstracting functionality is also included in the next higher layer.

  In the application we present in this paper, the Fusion Module provides easier access to information delivered from our user positioning systems. In this application, the Fusion Module only facilitates access to context information and does not change any information.

- The **Context Filter** is similar to the Primary Filter in that it filters information coming from the Fusion Module. However, the Context Filter acts on a higher level and is therefore able to perform "intelligent" filtering. For example, a positioning system may send periodic events indicating which user is in which room. The Context Filter takes these events and is able to generate higher-order events from them, such as "person X entered room Y".

- The **Interpreter** contains logic for implying higher-level context from the information supplied by the Context Filter. Its task is to derive context events which cannot be directly measured. Consider a situation where there are many people in a room, the noise level indicates people speaking, and the lights are turned on. None of the individual context "readings" alone is sufficient to infer that a meeting is taking place, but put together, the Interpreter can deliver this result.

- The **Application** layer contains the application using our context stack.

3 Implementation

Our implementation of the multi-device browsing architecture has three main components: a number of clients running a customized browser, the dialog manager, and a sensor infrastructure that detects which clients are in the vicinity of the user.

3.1 Browser Client

The most important task of any client in such a "virtual browser" is to render its respective part of the web application. This aspect of the client is no different from any normal single-device web browser. However, the client must perform a few additional tasks that current web browsers are not capable of without modification.

Recall our goal that roaming from one device to another should not require anything more than walking from the first device to the second. Previously, this type of roaming has been implemented, for example, by forwarding a user's X-Window desktop to whatever desktop PC is nearest to her, and is also referred to as *Teleporting*[6]. To achieve teleporting of a browser window, the dialog manager must be able to tell the browser on a device to start displaying a page as soon as the user walks up to that device. So our first extension of the browser is that it must allow the dialog manager to remotely open or close ("push") a browser window with a certain page.

Another problem is that normal web forms transmit the state of the form elements (text boxes, radio buttons, etc.) to the server only when the user presses the submit button. If we consider multiple devices to form a "virtual browser", then the state of form elements needs to be synchronized, to ensure that the state shown to the user is consistent across all devices. Requiring the user to press a button to synchronize the views would be contrary to our goal that the interaction should be effortless. Therefore, the second new feature of the browser is to both send status updates to and receive them from the dialog manager whenever the state of any form element changes.

The same applies for the focus (i.e., where the cursor is) across several views. For purely visual clients, this may seem trivial, but it is important when a speech client is added to a visual client for multimodality. When making speech input, the user will expect that her input ends up in the form field that visibly has the focus. However, the *visible* focus is displayed by a different client than the speech client. Therefore, focus must be synchronized across clients as well.

Our first implementation consists of a wrapper around the Internet Explorer, which communicates with the dialog manager and sends and receives the required updates whenever the user makes an input to the wrapped browser. We do not use any features exclusive to the Internet Explorer, and it is possible to implement a similar client for other browsers.

As a speech client, we are currently implementing a similar wrapper around IBM ViaVoice. We have also implemented clients for cell phones based on the Java Mobile Information Device Profile (MIDP) 2.0, and PDAs running a Java 2 Micro Edition (J2ME) Virtual Machine. We are currently working on a browser independent client based on a Java applet.

3.2 Dialog Manager

As described above, the dialog manager is the server that receives all the updates from the various clients. Whenever an input occurs at one of the clients, the client sends this input to the dialog manager, which in turn relays this update to all the other clients.

In our implementation, the dialog manager is simply a Java program with an event loop. Updates that are received from the clients are enqueued at the end of the event loop and processed in the order of their arrival. One other type of event that the dialog manager has to process are the events that occur whenever the user enters or leaves a device (see Section 3.3). Whenever such events occur, the dialog manager calls the transcoder in order to generate web pages adapted to the new set of devices, and pushes the outcome of this transcoding to the devices.

When speaking about a “server” in the context of web applications, it is important to point out that the dialog manager is not the same server as the one that receives and processes user input when the user is done and submits a form. The dialog manager is only responsible for synchronizing multiple clients in a “virtual browser”, and therefore is part of the browser, not the application server. It is even preferable to have the dialog manager running on a device near the user, for example one of the client devices, to ensure that feedback to input is shown with low latency.

However, we are also investigating applications beyond traditional web applications, and allow custom Java applications to connect directly to the dialog manager. These applications are notified of any input that takes place, and can react to it immediately, without waiting for the user to finish her input and press the submit button. In this context, the dialog manager acts like a widget toolkit that is able to place its widgets across several devices.

3.3 Sensors and Context Processing

Since we rely on tracking which devices are near the user, we need cheap and unobtrusive sensors for tracking them. We use a system consisting of three distinct devices: tags, badges, and room receivers (see Figure 3). Badges and room receivers are similar to the Active Badge location system[7,8]. The novel part of our system are the tags, which allow identifying objects (like displays and other means of interaction) in the user's field of view.

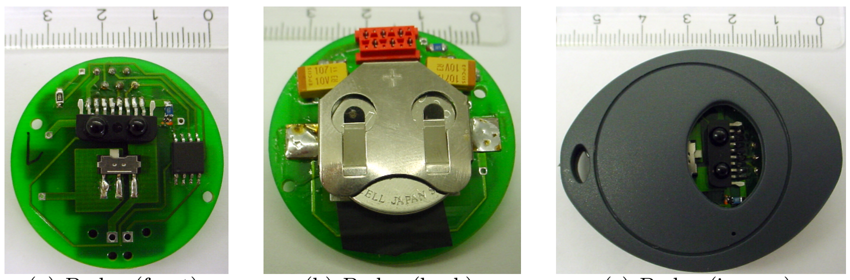
(a) Badge (front). (b) Badge (back). (c) Badge (in case).

Figure 3. A small badge. Tags use the same hardware but different software. They also draw power from a USB cable (not shown) instead of a battery. The scale on the pictures shows centimeters.

- **Tags** (or device tags) are small infrared emitters that we can attach to both fixed and mobile devices. A tag has a unique *tag identifier* which it broadcasts periodically. These broadcasts can only be received by nearby (ca. 1.2m) receivers within direct line of sight. Therefore reception of a tag ID conveys the context that the receiver is close to and facing the tagged object. Tags do not receive any transmissions.

- **Badges** are intended to be worn by people. Badges both transmit and receive through their infrared interface. Each badge has a unique *badge identifier* which it broadcasts periodically. This transmission uses an error resilient encoding and can be received several meters away, even if the signal propagates indirectly by reflection. Reception of a badge ID conveys the context that the wearer is in the same room as the receiver.

  Badges also listen for tag ID broadcasts. If a badge sees a tag it appends the tag ID to its next badge ID broadcast. Such a message conveys the context that the badge's wearer is in the same room as the receiver, and is nearby and facing a tagged object.

- **Room Receivers** are firmly installed in a room, listen for badge broadcasts, and relay them to the local network. They insert the room identifier of the room they are installed in into the message. There are two distinct messages types: "badge `b` seen in room `r` facing object `t`", and "badge `b` seen in room `r` not facing any tagged object".

Figure 4 shows the sequence of events when a user enters the proximity of a tagged device. The user's badge continuously broadcasts its identifier to the room receiver (step ①). When he approaches a tagged device in step ②, his badge starts to receive the tag's identifier (step ③). In step ④ the badge relays this identifier together with its own to the room receiver, which passes it on to the network for further processing in the context stack.

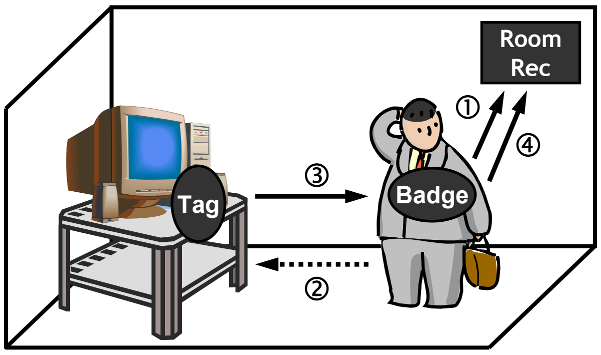

Figure 4. Sequence of events in associating a device.

A network service, which corresponds to the primary filter and fusion module in our context stack, filters out events from users we have no interest in, and maps the IDs to more meaningful forms of identity (badge IDs to user names and tag IDs to device names and network addresses).

After this first filter, we still have a stream of "person X in room R" or "person X in vicinity of device D" events, which repeat once per second. However, what we are interested in are "person X *entered* vicinity of device D" and "person X *left* device D" events when this stream starts and ends. This filtering is done at the context filter level. This filter also adds error resilience. It does not send out the "enter" event until several pings have been received, and delays the "left" event until several pings have been missing. This filters out transmission errors and occasional lost packets in the infrared communication.

One of the main advantages of using a context stack is that it abstracts from the underlying sensors. Therefore we are able to employ other sensors than the badge system for tracking the user and her devices. For example, we use a second location system called IRIS[9]. It has higher accuracy than the badge system, but is more difficult to set up.

The use of mobile devices, which are not tracked by either location system, is detected through a simple heuristic. We can sense whether they currently enjoy their owner's attention by querying whether they are turned on. This heuristic works fairly well, because most mobile devices quickly go to standby when not used for power saving reasons.

Aggregating the context information that comes from the different sensors is currently handled by the dialog manager, which corresponds to the application level in our context stack. Strictly speaking, this kind of functionality belongs in the interpreter level, but for efficiency reasons, we have combined the two levels in the dialog manager.

The system described in this paper is only one example for using our context stack. For more complex examples of how the general context stack can be used in other kinds of applications, please see Austaller et al.[5].

4 Related Work

4.1 Multiple-Device Interfaces and Single Authoring

The combined use of portable and stationary devices has been explored before (e.g. by Rekimoto[10] or Myers[11]). However, such research usually focused on one fixed combination of devices. We try to utilize whatever set of devices happens to be convenient for and available to the user depending on the context, not a single fixed set of devices.

The first attempts to use a browser to access multiple devices at once was first investigated for remote controlling other associated devices from one central device. The UbicompBrowser[12] controls nearby devices by clicking specialized URLs on a handheld device. For example, the URL `tv://local/station` switches a nearby TV to the channel "station". Interaction with this system always takes place through the handheld; associating nearby devices to render the web page that contains the links in higher quality was not among the goals of the UbicompBrowser.

The Small Screen / Composite Device (SS/CD) project[13] renders multimedia on multiple federated devices, which are automatically chosen from the devices available in the user's environment. The idea of federating devices and selecting those devices based on the user's context is similar to ours. However, rendering multimedia (such as a movie) is considerably different from rendering a web page or web application.

The development of a multimodal web browser by synchronizing browsers on multiple devices has been described by Kleindienst et al.[14]. However, this project considers a more or less fixed set of devices. Authoring for this synchronized browser is done by authoring separate versions of the same page in the native format for each device (HTML, VoiceXML, WAP) and glue code. This is similar X+V[15], proposed by Opera and IBM, which mixes XHTML, VoiceXML and ECMAScript in one document. In both cases the interface has to be authored more than once, and the author has to ensure consistency and operability manually.

Another project called WebSplitter[16] also synchronizes multiple browsers, but for a different reason: the browsers of different users are synchronized to achieve co-browsing (the navigation of several users is synchronized).

A project by Bandelloni and Paternò synchronizes interaction between exactly two devices[17]. The interface is split up into a control part and a visualization part. The visualization part is allowed to migrate, while the control part is not. This system uses task models to author user interfaces. Since these task models are device-independent, the visualization part could migrate among different types of devices.

While our work only migrates the user interface and keeps the application logic on a server, one might also consider migrating the whole application. The Roam framework[18] does this. It is a Java toolkit which allows writing applications as a set of migratable components. Each component can either be written using a device independent widget toolkit, or as multiple device dependent versions. At runtime, Roam chooses the appropriate device dependent version, or if none exists the device independent version.

4.2 Context Awareness

In the beginning of context-aware computing, exploring the possibilities and advantages to users of this new kind of application has been at the focus of research like the phone forwarding application built on top of the Active Badge Location System[8] and the GUIDE project[19,20].

With the increasing mobility of devices, work such as the dissertations of Schilit[21] or Nelson[22] explored the efficient distribution of notifications to clients. Dey introduced the separation of concerns and reusage in the design process of context aware systems[23]. Later models include work by Schmidt and Gellersen[24], and Grossmann et al.[25]. Recent work has studied automatic building of context models[26] and predicting context[27].

In contrast to previous work, our context model explicitly formalizes the different layers between the context sources and the application. Formalizing the context model in layers gives us the advantage that we can use existing modules for each layer and plug in the appropriate modules or layers on-demand. Furthermore, our layered model, based on ubiquitous services, is highly scalable, and can handle a large number of context sources, filter, and context consumers (applications) concurrently.

The more refined context models have also advanced the development of new kinds of context, although location still remains the most important context information. There are many positioning systems which can act as a source of location, including infrared[8], cell based WLAN systems[20], and ultrasonic[28,29]. Some systems use RFIDs[30] or triangulation of WLAN signal strengths from multiple base stations[31] as a location system.

5 Summary

In this paper we have addressed the problem of accessing web applications through the "countless appliances pervading the workspace" paradigm of ubiquitous computing, rather than a full-featured desktop PC. We have presented our architecture which allows seamless roaming between devices. Our architecture is integrated with our context stack implementation, which provides the information which devices are available. The dialog manager performs the splitting of the interface, any required transcoding, and synchronizes the views on different devices. We have presented an implementation of our architecture, using Internet Explorer and our infrared badges, tags, and room receivers which act as location information sources.

References

1. Ahmed Seffah and Homa Javahery, editors. *Multiple User Interfaces, Cross-Platform Applications and Context-Aware Interfaces.* John Wiley & Sons, Ltd, January 2004.
2. Mark Weiser. The Computer for the 21st Century. *Scientific American*, 265(3):66–75, September 1991.
3. Elmar Braun, Andreas Hartl, Jussi Kangasharju, and Max Mühlhäuser. Single Authoring for Multi-Device Interfaces. In *Adjunct Proceedings of the 8th ERCIM Workshop "User Interfaces For All"*, Vienna, Austria, June 2004. `http://www.ui4all.gr/workshop2004/publications/adjunct-proceedings.html`.
4. Elmar Braun and Max Mühlhäuser. Extending XML UIDLs for Multi-Device Scenarios. In Kris Luyten, Marc Abrams, Jean Vanderdonckt, and Quentin Limbourg, editors, *Proceedings of the AVI2004 Workshop "Developing User Interfaces with XML: Advances on User Interface Description Languages"*, pages 143–149, Gallipolli, Italy, May 2004.
5. Gerhard Austaller, Jussi Kangasharju, and Max Mühlhäuser. Using Web Services to build Context-Aware Applications in Ubiquitous Computing. In Piero Fraternali, Nora Koch, and Martin Wirsing, editors, *Lecture Notes in Computer Science 3140, Web Engineering: 4th International Conference, ICWE 2004*, pages 483–487, Munich, Germany, July 2004.
6. Frazer Bennett, T. Richardson, and Andy Harter. Teleporting - Making Applications Mobile. In *Proceedings of 1994 Workshop on Mobile Computing Systems and Applications*, Santa Cruz, USA, December 1994.
7. Andy Harter and Andy Hopper. A Distributed Location System for the Active Office. *IEEE Network*, 8(1):62–70, 1994.
8. Roy Want, Andy Hopper, Veronica Falcão, and Jonathan Gibbons. The Active Badge Location System. *ACM Transactions on Information Systems*, 10(1):91–102, 1992.
9. Erwin Aitenbichler and Max Mühlhäuser. An IR Local Positioning System for Smart Items and Devices. In *Proceedings of the 23rd IEEE International Conference on Distributed Computing Systems Workshops, 3rd International Workshop on Smart Appliances and Wearable Computing (IWSAWC03)*, pages 334–339, Providence, USA, 2003. IEEE CS Press.
10. Jun Rekimoto. A Multiple Device Approach for Supporting Whiteboard-Based Interactions. In *Proceedings of ACM CHI 98 Conference on Hu-*

man Factors in Computing Systems, pages 344–351, Los Angeles, USA, 1998. ACM Press.

11. Brad A. Myers. Using Handhelds and PCs Together. *Communications of the ACM*, 44(11):34–41, 2001.
12. Michael Beigl, Albrecht Schmidt, Markus Lauff, and Hans-Werner Gellersen. The UbicompBrowser. In Constantine Stephanidis and Annika Waern, editors, *Proceedings of the 4th ERCIM Workshop on 'User Interfaces for All'*, Stockholm, Sweden, October 1998.
13. Thai-Lai Pham, Georg Schneider, and Stuart Goose. A Situated Computing Framework for Mobile and Ubiquitous Multimedia Access Using Small Screen and Composite Devices. In *Proceedings of the 8th ACM international conference on Multimedia*, pages 323–331, Marina del Rey, USA, 2000. ACM Press.
14. Jan Kleindienst, Ladislav Seredi, Pekka Kapanen, and Janne Bergman. Loosely-coupled approach towards multi-modal browsing. *Universal Access in the Information Society*, 2(2):173–188, June 2003.
15. Jonny Axelsson, Chris Cross, Håkon W. Lie, Gerald McCobb, T. V. Raman, and Les Wilson. XHTML+Voice Profile 1.0. W3C Note, December 2001. `http://www.w3.org/TR/xhtml+voice/`.
16. Richard Han, Veronique Perret, and Mahmoud Naghshineh. WebSplitter: A Unified XML Framework for Multi-Device Collaborative Web Browsing. In *Proceedings of the ACM CSCW'00 Conference on Computer-Supported Cooperative Work*, pages 221–230, Philadelphia, USA, December 2000. ACM Press.
17. Renata Bandelloni and Fabio Paternò. Flexible Interface Migration. In *Proceedings of the 9th International Conference on Intelligent User Interfaces*, pages 148–155, Funchal, Madeira, Portugal, 2004. ACM Press.
18. Hao-hua Chu, Henry Song, Candy Wong, Shoji Kurakake, and Masaji Katagiri. Roam, a seamless application framework. *Journal of Systems and Software*, 69(3):209–226, January 2004.
19. Gregory D. Abowd, Christopher G. Atkeson, Jason Hong, Sue Long, Rob Kooper, and Mike Pinkerton. Cyberguide: a mobile context-aware tour guide. *ACM Wireless Networks*, 3(5):421–433, 1997.
20. Nigel Davies, Keith Mitchell, Keith Cheverst, and Gordon Blair. Developing a Context Sensitive Tourist Guide. In *Proceedings of the First Workshop on Human Computer Interaction for Mobile Devices*, pages 64–68, Glasgow, U.K., May 1998.
21. William Noah Schilit. *A System Architecture for Context-Aware Mobile Computing*. PhD thesis, Columbia University, New York, USA, 1995.
22. Giles John Nelson. *Context-Aware and Location Systems*. PhD thesis, University of Cambridge Computer Laboratory, January 1998.
23. Anind K. Dey. *Providing Architectural Support for Building Context-Aware Applications*. PhD thesis, Georgia Institute of Technology, November 2000.

24. Albrecht Schmidt and Hans-Werner Gellersen. Modell, Architektur und Plattform für Informationssysteme mit Kontextbezug. *Informatik Forschung und Entwicklung*, 16(4):213–224, 2001.
25. Matthias Grossmann, Alexander Leonhardi, Bernhard Mitschang, and Kurt Rothermel. A World Model for Location-Aware Systems. *Informatik/Informatique, Zeitschrift der schweizerischen Informatikorganisationen*, 8(5):22–25, 2001.
26. Toshihiro Takada, Satoshi Kurihara, Toshio Hirotsu, and Toshiharu Sugawara. Proximity Mining: Finding Proximity using Sensor Data History. In *Proceedings of the Fifth IEEE Workshop on Mobile Computing Systems and Applications (WCSMA 2003)*, pages 129–138, Monterey, California, October 2003. IEEE Press.
27. Jan Petzold, Faruk Bagci, Wolfgang Trumler, and Theo Ungerer. The State Predictor Method for Context Prediction. In *UbiComp 2003: Adjunct Proceedings*, pages 191–192, Seattle, USA, October 2003.
28. Andy Ward, Alan Jones, and Andy Hopper. A New Location Technique for the Active Office. *IEEE Personnel Communications*, 4(5):42–47, October 1997.
29. Nissanka B. Priyantha, Anit Chakraborty, and Hari Balakrishnan. The Cricket Location-Support System. In *Proceedings of the 6th ACM International Conference on Mobile Computing and Networking (ACM MOBICOM)*, pages 32–43, Boston, USA, August 2000. ACM Press.
30. Lionel M. Ni, Yunhao Liu, Yiu Cho Lau, and Abhishek P. Patil. LANDMARC: Indoor Location Sensing Using Active RFID. In *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications (PerCom'03)*, pages 407–415, 2003.
31. Tomoaki Ogawa, Shuichi Yoshino, Masashi Shimizu, and Hirohito Suda. A new in-door location detection method adopting learning algorithms. In *First IEEE International Conference on Pervasive Computing and Communications (PerCom'03)*, pages 525–530. IEEE Computer Society, 2003.