

Adapting the Web Interface: An Adaptive Web Browser¹

K. Henricksen and J. Indulska

Department of Computer Science and Electrical Engineering, The University of Queensland

Email: {karen, jaga}@csee.uq.edu.au

Abstract

The growing number of mobile computing devices with diverse characteristics creates a requirement for seamless (device independent) access to computing resources of distributed systems. One of the most common applications in distributed systems is the Web browser, which is not only used to access resources on the Internet but also as an interface to many Information Systems applications. In this paper, we address types of adaptation that can be applied to a Web browser in response to diverse context changes, including changes in available computing resources, input and output device capabilities, network characteristics, location and user context. We also present a design and implementation of a Web browser that adapts to changes in its network and computing environment by exploiting context metadata.

1. Introduction

Most current applications and protocols for the Web are designed with traditional computing environments in mind, in which applications are run on stationary machines with access to wired networks. However, users are increasingly turning to mobile devices, which, unlike stationary devices, are typically characterised by resource-poor environments and frequently changing context. Thus, the next generation of applications and protocols will need to take into account the requirements of mobile and ubiquitous computing. They will need to be able to respond to users changing computing devices and mobility of computers and applications. Moreover, they will be required to provide services that exploit an awareness of the user's experience and current activities, as well as other aspects of context, such as location. In order to meet these requirements, applications must be capable of dynamically adapting to their environments.

Currently available Web browsers provide very little support for adaptation. Typically, they permit some types of static adaptation using configuration options, which

allow the user to manipulate user interface attributes including fonts, colours, and the loading of images, as well as behavioural aspects, such as caching strategies and the use of proxies. However, they do not support dynamic response to changes in context. This paper describes how Web browsers can employ sophisticated adaptation mechanisms to provide context-aware behaviour and user interfaces.

The structure of the paper is as follows. Section 2 describes a wide range of adaptation types that can be applied to a Web browser. Section 3 presents our design of an adaptive browser, Section 4 discusses a prototype that implements the design, and Section 5 briefly evaluates the prototype. Section 6 characterises related work. Finally, section 7 presents concluding remarks, including areas for future research.

2. Adaptation

Within this paper, the term adaptation refers to the alteration of an application's behaviour or interfaces in response to arbitrary context changes.

The types of adaptation that can be employed by an application depend on the nature of the application and the resources it requires. Adaptation mechanisms can be classified according to the types of context information they exploit. Classes of adaptation that can be employed by a Web browser include, but are not limited to those listed in Table 1. The remainder of this section describes adaptation mechanisms that fall into each of the adaptation classes.

2.1 Computing adaptation

Computing adaptation responds to the availability of resources such as CPU cycles and memory (both primary and secondary). Adaptation strategies in this class typically involve matching the consumption of resources to their availability. The availability of computing resources varies between devices as well as over time as

¹ The work reported in this paper has been funded in part by the Co-operative Research Centre Program through the Department of Industry, Science & Tourism of the Commonwealth Government of Australia.

Table 1. Adaptation classes

Adaptation Type	Description
<i>Computing</i>	Responds to the availability of computing resources, including CPU cycles and primary and secondary memory
<i>Communication</i>	Addresses changes in network resources, including disconnections and changes in bandwidth, latency and jitter
<i>Input/Output</i>	Responds to the availability and characteristics of input and output devices
<i>User</i>	Concerned with user experience and capabilities, as well as user context such as activities in which the user is engaged
<i>Location</i>	Responds to pertinent location knowledge, including location of users and computing devices

the computing load changes.

A Web browser can respond to severely limited CPU capacity by disabling CPU-intensive tasks, such as the running of applets or animations. In some cases, CPU requirements can be traded against user satisfaction or fidelity of output, such that high fidelity output is provided when processing resources are plentiful, and low fidelity output is provided when processing resources are scarce [1].

The principal usage of memory (both primary and secondary) by a Web browser is caching. Therefore, adaptation to memory availability should incorporate adaptation of the caching strategy. The period for which files are hoarded in the cache and the size of the cache are two attributes that can reflect the availability of memory.

Memory availability can also be considered when selecting between several variants of a resource loaded from a Web server. When memory is very scarce, the variant requiring the least memory can be obtained (such as the thumbnail version of an image). In less extreme situations, a balance between fidelity and memory requirements can be achieved.

2.2 Communication adaptation

A Web browser relies heavily on network resources to obtain data from remote hosts. Because the availability of these resources can vary greatly, particularly when relying on wireless links, a browser should be capable of matching its behaviour to the characteristics of the network.

Bandwidth availability can be reflected in the selection of resource variants; compact variants may be given preference over larger variants when bandwidth is scarce. When bandwidth is severely limited, bandwidth-intensive resources, such as video, audio and images, may be disabled altogether, or transformed to more compact representations prior to transmission over bandwidth-constrained links.

Jitter is primarily of concern with video and audio streams. Problems caused by jitter can be overcome by

employing adaptive caching strategies that take into account the current jitter rate.

2.3 Input/Output adaptation

With browsing becoming increasingly ubiquitous, the range of input and output modalities browsers need to support is increasing. Voice-based browsing enables users to browse over ordinary telephones and while engaged in other tasks, such as driving a car. Mobile phones and PDAs both enable screen-based browsing, but introduce constraints, including limited input capabilities and small screens, that require the provision of different types of user interface to those found on desktop computers. In the future other types of browsing will become possible with the availability of new input and output devices.

In order to support a range of device types with varying input and output capabilities, as well as different modes of interaction depending on the user's capabilities, preferences and activities, it is necessary for the browser to be capable of dynamically adapting its interface to the context.

The browser should be capable of presenting GUIs that are adapted to screen size and input devices, as well as less traditional interfaces, such as voice-based interfaces.

In addition, browsers should adapt the files they retrieve and present to users according to the output device characteristics. In the presence of a very small display, a browser may load the thumbnail version of an image, whereas when only audio output is available, the browser may obtain an audio description of the image.

2.4 User adaptation

User adaptation contrasts with the adaptation classes described in the previous sections as it focuses on context related to the application user, rather than to the hardware and software resources. It takes into account factors such as the user's level of experience with the application, the user's capabilities and preferences, and other aspects of

the user's context, including the activities in which the user is participating and the other participants. Adaptation to user context addresses the diverse requirements of different users, as well as the changing requirements of a single user over time.

Knowledge of user history can be exploited to provide a user interface tailored according to the user's experience level and commonly performed tasks. Novice users may be presented with user interfaces that prominently display fundamental application functionality, with sophisticated functions largely hidden. As users become more experienced, increasingly sophisticated functionality may be revealed. Additionally, the set of available functions may be tailored according to the user's history. If a user frequently executes a particular command sequence, that sequence may become available as an atomic action.

Capabilities, such as a user's ability to understand both English and French, and preferences, such as a user's preference for English over French, may be reflected in both the user interface and behaviour of the browser. Additionally, preferences and capabilities, along with other user context, can be employed as browsing parameters, enabling information loaded by the browser to be context-dependent.

2.5 Location adaptation

Knowledge of the location of users, devices and other objects can be used within the Web to provide location-aware services and searches. Various location-sensitive services have already been developed, such as GUIDE, a context-sensitive tourist guide [2].

Browsers can also adapt according to location. For instance, a browser's security policies may be influenced by knowledge about the local security domain. A browser may choose to transmit sensitive information only when it operates in a trusted domain, such as behind a company firewall, and may also suppress sensitive information when people who are not authorised to access the information are nearby.

3. Design

The types of adaptation that have been described are all dynamic in nature; that is, they are reactions to dynamic changes in context. There are two distinct issues involved in creating applications that are dynamically adaptive: creation of adaptation mechanisms for responding to context changes, and detection of the context changes that lead to adaptation. In this paper, we are concerned with the first issue.

This section presents the design of an adaptive Web browser and server, focusing on a limited subset of the adaptation types described in the previous section. The types of adaptation we have chosen to address fall into the communication and input/output adaptation classes. We

first describe ways in which HTTP can be extended to enable HTTP clients and servers to cooperate in order to achieve adaptation. Next, we discuss the impact of the protocol modifications on the design of the Web server, as well as the incorporation of adaptation mechanisms into the browser. Finally, we briefly characterise our approach to context monitoring.

3.1 HTTP Support for Adaptation

The adaptation mechanisms employed by Web browsers can be either solely the concern of the browser or require the cooperation of multiple parties. For example, a Web browser that adapts its user interface to the display type by changing the widgets used within the interface can achieve this adaptation independently. However, if the browser supports adaptation to network changes, the Web servers with which the browser communicates should also participate in the adaptation. This section characterises our approach to achieving the latter type of adaptation.

In order to realise adaptation that involves cooperation, it is necessary for the cooperating parties to be capable of communicating their requirements to each other. This can be achieved either by creating a communication protocol expressly for this purpose, or by extending one or more existing Web protocols. We favour the second approach due to its relative simplicity. The remainder of this section will concentrate on extending HTTP/1.1 [3]. It should be noted that other protocols employed in the Web, such as FTP, could similarly be extended.

HTTP assumes that there are two communicating parties, one of which plays the role of the server and the other the client. The client requests an operation, such as GET or PUT, from the server using a request message. A unique Uniform Resource Identifier (URI) indicates the resource to which the request applies. Additional information pertaining to the request, such as the content types, languages and encodings that will be accepted by the client, authorisation information and caching information, can be supplied using request header fields.

A server responds to a request with a response message, which includes the status of the request (e.g. OK, Forbidden, Not Found), header fields containing additional information, and possibly a message body containing data requested by the client.

There are many ways in which HTTP/1.1 can be augmented to provide support for adaptation involving cooperating parties. The extended protocol could allow the HTTP client to communicate its context parameters within request messages, permitting the server to adapt its behaviour to the client's circumstances. Alternatively, the HTTP client could assume a more active role and transmit instructions directing the server about how to adapt. Other approaches are also possible. Only the first approach will be explored in this paper.

The client's context information can be conveyed to the HTTP server using a request header field, such as the following:

Context: Bandwidth low; Display 1280 * 1024, 16 bit colour

Here we assume a model for specifying context that is based around context attributes that have associated value lists of zero or more values. An alternative context specification model based on CC/PP [4], which is currently being developed by the W3C, could be used in the future.

The context field shown above indicates that the client has limited bandwidth and a display that is 1280 pixels wide and 1024 pixels high with 16-bit colour depth. If necessary, the server can relate its context information to the client by including a similar header field in its response message.

3.2 Adaptive Web Server

The HTTP server should take advantage of the context information supplied by the client to adapt its response to match the client's circumstances. If the client indicates a low level of bandwidth, the server might respond by sending compressed files to the client. Similarly, if the client has a display with low colour depth, the server might send images that use few colours.

HTTP/1.1 provides a mechanism by which several alternative copies of a resource, known as resource variants, can be associated with a single URI. This facility is already used by some HTTP servers to adapt their responses to preferences of the client. The algorithm that the server uses to determine which variant is most appropriate is known as a remote variant selection algorithm (it should be noted that there are also client-side algorithms that perform the selection). Several such algorithms have been defined. One of these is RVSA/1.0, which is described by Internet RFC 2296 [5]. The algorithm computes quality values for each available variant and defines the best variant as the one with the highest value (or the first of several such variants). Quality values are determined as follows:

$$Q = \text{Round5}(qs \times qt \times qc \times ql \times qf)$$

The q -values are quality indices, ranging from 0 to 1, that assess the suitability of the variant in particular areas. They are interpreted as follows:

- qs reflects the fidelity of the variant
- qt indicates the suitability of the variant's media type according to the user's media-type preferences
- qc indicates the suitability of the variant's character set according to the user's character set preferences
- ql indicates the suitability of the variant's languages according to the user's language preferences
- qf is the features quality factor, which reflects the suitability of the variant according to additional

features specified by the user within the Accept-Features request header

The algorithm currently enables adaptation of server responses to changes in the preferences of the user. We propose extensions to the algorithm to allow it to additionally support adaptation to changes in context. To achieve this, we incorporate an extra index, qe , into the variant quality computation, which reflects the suitability of the variant according to the current environment. The information used to compute this index is extracted from the context request header field that we defined in the previous section. The modified quality value computation is shown below.

$$Q = \text{Round5}(qs \times qt \times qc \times ql \times qf \times qe)$$

We define qe as the product of factors computed separately for each of the attributes specified by the client in the context header. As an example, suppose that the user specifies the context header below.

Context: Bandwidth low; Display 1280 * 1024, 16 bit colour

In this case, qe is the product of:

- a factor that describes the variant's suitability for a low level of bandwidth, and
- a factor describing how appropriate the variant is for the user's display type.

The computation of each of the component factors is defined in a manner that is appropriate to the type of context concerned. The remainder of this section describes, for illustrative purposes, one way in which bandwidth factors can be computed. The bandwidth factor, qb , reflects how well suited the variant is to the current level of bandwidth. We assume for simplicity that bandwidth is characterised by the user as low, medium or high. qb is computed as follows:

- If bandwidth characterised as medium, then qb is $\frac{2}{3} + \frac{\text{size of smallest variant}}{3 * \text{size of variant}}$
- Otherwise, if bandwidth is characterised as low, then qb is $\frac{\text{size of smallest variant}}{\text{size of variant}}$
- Otherwise, qb is 1

In other words, when the bandwidth level is low or medium, the suitability of a variant is regarded as being inversely proportional to its size. However, when the bandwidth level is medium, the bandwidth factors are scaled so that they have less influence on overall variant quality values. In the remaining case, when the bandwidth level is high or unknown, the bandwidth factors do not contribute to the variant selection process.

We illustrate our modified variant selection procedure using the variants shown in Figures 1 - 3. In Table 2, we show the values for qb and Q for each of the variants. The qb factors are computed according to the algorithm above. The Q values represent the overall quality values, assuming that variants 1, 2 and 3 are assigned fidelity (qs)



Figure 1. High fidelity GIF image (16293 bytes)



Figure 2. Low fidelity JPEG image (2584 bytes)

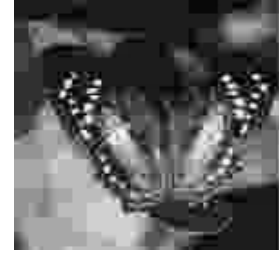


Figure 3. Low fidelity, size-reduced JPEG image (1507 bytes)

values of 1, 0.5 and 0.4 respectively. We further assume that all q -values other than qs and qb are equal to 1 for all of the variants.

According to our modified version of RVSA/1.0, variant 1 is best in situations of high or medium bandwidth, and variant 3 in those of low bandwidth.

3.3 Adaptive Web Browser

This section characterises our design of an adaptive Web browser. The set of adaptation mechanisms we incorporate forms a subset of the set of adaptation types listed in Section 2.

We propose the following three general classes of adaptation:

- **Adaptation to display type:** The browser adapts the interface it presents to the user according to the type of the display that is used. In particular, fonts and widgets are adjusted to match the size of the screen. Additionally, the browser conveys information about the display to HTTP servers using the context header field, enabling the servers to adapt their responses accordingly. If the display type changes at run-time, the browser dynamically alters its interface and behaviour in response.
- **Adaptation to bandwidth availability:** The Web browser communicates to the server the amount of bandwidth that is available along its network link to the server, if this information is available. The server uses this information to adapt its responses accordingly.

Unfortunately, it is infeasible for the browser to have knowledge of the bandwidth along every link of the network. In reality, the browser typically only has reliable information about links connecting it to the servers that were targets of the preceding few requests. However, in practice, the user will often send several requests to the same server in quick succession, so this approach remains useful despite its limitations.

- **Adaptation to network disconnections:** The browser responds to the severing of a link to a HTTP server by using mirror sites that remain connected, whenever possible. The browser maintains collections of mirror sites that can be edited by the user. It is anticipated that in the future mirror information could be obtained in an automated fashion by querying HTTP servers and caching the information for use when a disconnection arises.

All types of adaptation that we have described are dynamic; that is, they occur when the browser is first invoked, as well as when the browser's environment changes. Each of the adaptation mechanisms can be enabled and disabled according to the user's preference.

3.4 Context Specification and Monitoring

In order to achieve dynamic adaptation, we require a monitor that tracks the state of the browser's environment, and triggers adaptation when significant changes occur. While the monitoring function could be incorporated directly into the browser, we believe this is

Table 2. Bandwidth and overall quality factors for the three image variants

Bandwidth Level	qb			Q		
	Variant 1	Variant 2	Variant 3	Variant 1	Variant 2	Variant 3
High	1	1	1	1	0.5	0.4
Medium	0.69750	0.86107	1	0.69750	0.43053	0.4
Low	0.09249	0.58320	1	0.09249	0.29160	0.4

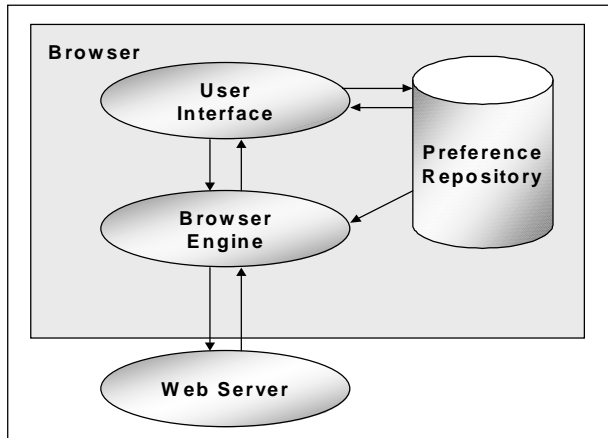


Figure 5. Original browser architecture

an undesirable solution. It leads to a complicated browser, as well as a significant duplication of effort if several applications need to monitor the same resources. Instead, we propose the use of a separate agent whose task is to track the environment and notify applications of significant context changes. Our general model of context specification for adaptive systems can be found in [6]. In this paper, the specification and monitoring of context are not our primary concerns; however, for the purpose of building a prototype, we have designed a simple resource-monitoring agent that satisfies the requirements of our adaptive browser.

Clients of the monitor receive notification of context changes through events. The clients register interest in certain types of event by providing predicates describing the types of changes they consider relevant. They also provide a notification interface into which relevant events are pushed by the monitor.

The predicate types that we have defined include mathematical comparison predicates ($<$, \leq , $>$, \geq and $=$), *changed*, which is true every time the context value is altered, and *range*, which is true whenever the context value changes so that it falls between two specified bounds. All of the predicate types assume that the state of a resource can be described by a single value.

A predicate may include a parameter that provides information affecting its interpretation. For example, a predicate related to bandwidth may be parameterised by the name of the link over which the bandwidth should be monitored.

The following are example predicates:

Notify when bandwidth $<$ 2 Mbps to dstc.edu.au
 Notify when display has changed

4. Implementation

In order to test our design, we have constructed a

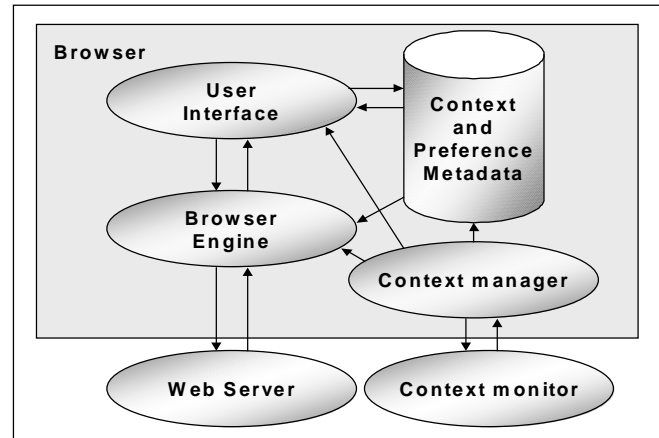


Figure 6. Architecture of the adaptive browser

prototype using Python. This section describes the three components of our prototype: the adaptive Web browser, adaptive HTTP server and monitoring agent.

4.1 Adaptive Web browser

Our adaptive Web browser is based on Grail, an extensible browser created by the Corporation for National Research Initiatives (CNRI), for which the source code is freely available. A simplified representation of the browser's architecture is shown in Figure 5. The browser has three main components: the user interface, browser engine and preference repository.

The user interface component is responsible for providing a GUI built using the Tk toolkit. It displays both the browsing interface, consisting of menus and toolbars, and the information loaded by the browser (Web pages and applets). The user interface also translates input from the user into operations provided by the second component, the browser engine.

The browser engine provides the core functionality of the browser. It performs communication with Web servers using a variety of protocols, including HTTP and FTP, and is also responsible for caching.

The preference repository is a data store containing a dictionary of preference attributes (such as cache size and font size) and associated values. The values are obtained through the user interface, and used by both the user interface and browser engine to provide customised behaviour.

We have extended the browser to support context awareness and adaptation. The design of the extended browser is shown in Figure 6.

The required changes have been threefold. First, we have added a context manager, which enables context awareness within the browser. This manager is responsible for registering interest with the monitoring agent in certain types of context event and handling event

notifications. The latter task involves the propagation of context information to the metadata store and the invocation of appropriate adaptation mechanisms in the user interface and browser engine components.

The second modification involved the extension of the preference repository to support additional types of browser metadata, namely context information supplied by the monitoring agent.

Finally, the user interface and browser engine components were augmented to support the types of adaptation outlined in Section 3.3.

We have been careful to ensure that the adaptive browser can be easily extended to support new adaptation policies that may be required in the future. The support for extension is twofold:

- The design of the context manager component incorporates plug-in modules in order to facilitate the addition of arbitrary types of context adaptation.
- The decoupling of the user interface from the core functionality of the browser makes it relatively easy to substitute entirely types of user interface in the future. We envisage the use of dynamic adaptation between entirely different types of interfaces, for example, between a GUI and a voice-based interface.

4.2 Adaptive Web server

We have implemented a server that supports our extensions to HTTP/1.1 and RVSA/1.0. Our server extends the SimpleHTTPServer class provided by the Python library module of the same name.

Currently, the server supports adaptation to network bandwidth and the client's display type. As we envisage that our browser may support other types of adaptation in the future, we implement the support for each context type as a plug-in module. The plug-in encapsulates the interpretation of context information and the computation of quality factors for variants.

4.3 Monitoring agent

Although the monitoring of context is not our primary interest, we have developed a monitoring agent that supplies the browser with information about its environment. The monitor currently supports tracking of the display type, as well as monitoring of the bandwidth and connection status of network links specified by the client. Monitoring of the display is performed using the Unix `xdpiinfo` utility, while the network is monitored using `bing`, a program based on ping which estimates the throughput on a link by measuring the roundtrip times for ICMP echo requests. The monitor also provides a graphical interface through which the user can manipulate resource data for simulation purposes.

5. Evaluation

Through our own experiences with the browser we have verified that the adaptation mechanisms introduced into the prototype have generally improved its performance over the original version, particularly when considering download latency and optimisation of screen space. However, we have yet to perform usability trials to determine whether the improved performance translates into improved usability. The usability trials should address the following issues, among others:

- Does the decision to balance the bandwidth and display characteristics against the fidelity of variants improve the browsing experience?
- Is complete transparency of adaptation desirable, or does it instead make behaviour appear erratic?

6. Related Work

There have been several significant contributions in the area of adaptability. The Coda distributed file system [7], developed at Carnegie Mellon University, demonstrated the use of application-transparent adaptation in dealing with disconnected hosts. This research formed the basis for Odyssey [8], a set of extensions to NetBSD that provide support for application adaptation. Part of the work on Odyssey demonstrated how distillation could be used to support adaptation of information used by a Web browser. Information fetched by the Web browser was intercepted by a distillation server and transformed to the level of fidelity requested by the browser's agent, called the cellophane. All adaptation occurred externally and transparently to the browser. While this strategy has the advantage that it involves no modifications to the browser itself, it is more limited in the types of adaptation that can be achieved than our approach, in which the browser plays an active role in the adaptation.

Mobiware [9], developed at Columbia University, is a middleware toolkit that supports adaptation to varying network conditions. It provides an infrastructure of programmable network objects that can be manipulated to provide applications with their desired Quality of Service (QoS). Applications specify their QoS requirements through a QoS API, in the form of a utility function and an adaptation policy. The utility function expresses the level of satisfaction of the application with different levels of bandwidth, while the adaptation policy determines how the application's bandwidth allocation will vary as resource availability changes. Mobiware concentrates solely on network-related QoS.

Research at Xerox PARC has focussed on context-based adaptation of applications [10]. A framework has been developed to support this type of adaptation. This is based around the dynamic environment server, which

manages environment variables and delivers updates to clients who have subscribed to the server as the variables change. There is typically one server per context, where a context might be a room or work group.

In addition, some work has been carried out on the development of Web browsers for mobile devices, such as PDAs. These browsers include HandWEB, Palmscape and Top Gun Wingman [11], designed for the PalmPilot, and PocketWeb, for the Apple Newton MessagePad. PocketWeb and Top Gun Wingman are able to support limited adaptation by connecting to a special proxy, which can perform some transformations of HTML documents and images to make them more suitable for display on a PDA. The HotJava browser can also be used on various mobile computers. HotJava is a lightweight browser that can be statically configured for mobile devices through customisation of the user interface and installation of new content and protocol handlers. All of these browsers incorporate some static mechanisms for coping with a mobile environment, however, unlike our browser, none are capable of dynamic adaptation to changes in characteristics of the network and local machine. The adaptation mechanisms supported by our browser allow it not only to execute on a range of different computers, but also to cope with run-time context changes.

The DISCIPLE application framework [12], based on Java components, supports adaptation of XML documents used by collaborative applications. It provides transformation of documents according to client profiles, thus allowing collaborators to have different views of the same data depending on their resource availability, QoS requirements and personal preferences. The focus of the framework is on data adaptation, whereas we consider a broader set of adaptation types.

7. Concluding Remarks

In this paper, we have described types of adaptation that can be employed to allow a Web browser to respond to changes in context and presented our design and implementation of an adaptive Web browser. We have considered types of adaptation that can be carried out internally within the browser, as well as types that require the cooperation of the Web server. In the latter case, protocol support is required for the communication of context information between the cooperating parties. To meet this requirement, we have extended the HTTP/1.1 protocol with the context header. Moreover, through our extensions to RVSA/1.0, we have demonstrated how the information carried in the header can be used to influence the server's behaviour when faced with a choice between several variants.

We have built prototypes that currently support adaptation to bandwidth and display variations, as well as network disconnections. The adaptation mechanisms

generally increase performance, however, usability trials are required to determine whether the performance improvements in fact lead to enhanced usability.

8. References

- [1] Satyanarayanan, M. and Narayanan, D., "Multi-Fidelity Algorithms for Interactive Mobile Applications", *Proceedings 3rd Intl. Workshop on Discrete Algorithms and Methods in Mobile Computing and Communications*, Seattle, Washington, USA, August 1999.
- [2] Cheverst, K., Davies, N., Mitchell, K. and Friday, A., "The Role of Connectivity in Supporting Context-Sensitive Applications", *Proceedings 1st International Symposium on Handheld and Ubiquitous Computing*, Karlsruhe, Germany, September 1999.
- [3] Fielding, R. et al., "Hypertext Transfer Protocol -- HTTP/1.1", Internet RFC 2616, June 1999.
- [4] Reynolds, F. et al. (eds), "Composite Capability/Preference Profiles (CC/PP): Structure", W3C Working Draft, 21 July 2000.
- [5] Holtman, K. and Mutz, A., "HTTP Remote Variant Selection Algorithm -- RVSA/1.0", Internet RFC 2296, March 1998.
- [6] Bond, A., Gallagher, M. and Indulska, J., "An Information Model for Nomadic Environments", *Proc. 9th International Workshop on Database and Expert System Applications*, Vienna, Austria. IEEE Computer Society, pp 400 - 405, August 1998.
- [7] Satyanarayanan, M., "Mobile Information Access", *IEEE Personal Communications*, Vol. 3, No. 1, February 1996.
- [8] Noble, B. and Satyanarayanan, M., "Experience with adaptive mobile applications in Odyssey", *Mobile Networks and Applications*, Vol. 4, 1999.
- [9] Oguz A. et al., "The Mobiware Toolkit: Programmable Support for Adaptive Mobile Networking", *IEEE Personal Communications Magazine*, Special Issue on Adapting to Network and Client Variability, Vol. 5, No. 4, pages 32-44, August 1998.
- [10] Schilit, B., Adams, N. and Want, R., "Context-Aware Computing Applications", *Proc. Workshop on Mobile Computing Systems and Applications*, pp 85-90, Santa Cruz, USA, December 1994.
- [11] Fox, A. et al., "Experience With Top Gun Wingman: A Proxy-Based Graphical Web Browser for the USR PalmPilot", *Proc. IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware '98)*, Lake District, UK, September 1998.
- [12] Marsic, I., "A Software Framework for Collaborative Applications", *Proc. Collaborative Technologies Workshop*, Rochester, USA, November 1999.