Experiences in Developing Collaborative Applications Using the World Wide Web "Shell"

Andreas Girgensohn[1], Alison Lee[1], and Kevin Schlueter[1,2]

[1]NYNEX Science and Technology
500 Westchester Avenue
White Plains, NY 10604

[2]University of Toronto
Department of Computer Science
Toronto, Ontario M5S 1A4

{andreasg, alee, kevins}@nynexst.com

ABSTRACT

The components of the World Wide Web, which we call the World Wide Web Shell, provide a framework for collaborative application development in much the same way as an expert system shell does for expert system development. This development is quick enough to support rapid prototyping. Once the collaborative application is developed, the WWW Shell facilitates the distribution of the application and its data to geographically-separated users on diverse computing platforms. We have developed and deployed two collaborative applications, Design Intent and NYNEX Portholes, using the WWW Shell. These applications are described and our experiences developing them with the WWW Shell are detailed. In the process of developing these applications we discovered limitations of the WWW Shell which we present, along with suggested modifications and extensions to address them.

KEYWORDS: collaborative application, World Wide Web, rapid prototyping, HTTP server and clients, Portholes, Design Intent, forms and scripts, work groups, community of users, awareness and familiarization.

INTRODUCTION

The World Wide Web is often thought of as the latest and most user friendly way of providing information over the internet — the next step in the evolutionary line of telnet, ftp, and gopher. Although we concur with this, we argue that the components used to access and provide information on the Web can be used to build collaborative applications.

We will refer to the web components for building collaborative applications as the *World Wide Web Shell.* The term "shell" was chosen because the components provide basic functionality and services for developing collaborative applications in much the same way as an expert system shell provides components for developing expert system applications. Just as an expert system is built by adding information and knowledge specifications to an expert system shell, collaborative applications can be built by adding application data and a specification of application behavior (via HTML, CGI scripts and image files) to the WWW Shell.

We have implemented two diverse collaborative applications using the WWW Shell. The exploratory nature of the development effort requires the constant evolution of the application and data to respond to the needs of our users. We have found that the WWW Shell speeds up this process, because of its support for interface presentation and network transport. This facilitates, in a platform-independent manner, rapid prototyping and distribution of the application and ensures access to up-to-date application data. We can therefore obtain early and continuous feedback from users and other stakeholders.

In subsequent sections, we describe the two collaborative applications and place them in the context of some related work in hypermedia. We highlight their important features, discussing how the WWW Shell facilitated their development. We then summarize what we have learned by discussing the advantages and the limitations of the shell approach, along with ways in which these limitations could and are being overcome.

COLLABORATIVE APPLICATIONS

Our group is involved in developing methodologies and tools to support work groups (largely doing software development) in our organization. In order for groups to work effectively as a unit, communication among work group members is important. Communication facilitates information sharing that enables individuals to develop working relationships, to do work, to formulate a common understanding from divergent perspectives and to learn from each other. However, when team members are separated by time and/or space, these interactions can suffer. We are developing a number of collaborative applications to help support group communication. The applications are part of a communication infrastructure designed to:

Hypertext'96, Washington, D.C., USA

1. Promote socialization of the project team. The aim is for the team members to share information about themselves so that team members develop an awareness of and familiarization with other team members and to build channels for collaboration and coordination [6, 14, 19].
2. Capture, disseminate, and discuss project information and activities. The aim is to develop a project resource that enables team members to keep in touch with the project.
3. Provide a medium for communication. The aim is to enable group members to develop a common understanding through group discussion and brainstorming activities.

We will now briefly introduce two of the collaborative applications we have developed using the shell: Design Intent and NYNEX Portholes.

Design Intent

Design Intent is a set of resources for sharing and discussing project information and activities. It was first developed for a NYNEX project called Directory Assistance Direct Access Service (DADAS) [2]. A modified version has been developed for the NYNEX Telecommunity Project.

Design Intent contains a hypermedia document repository. An example of its content is a project team document containing information about who the team members are, how to contact them, and what part of the organization they come from (see Figure 1). Links exist within this page to team member documents containing information about an individual's contact information, photo, project role, and message of the day. This enables others to associate a team member's name to a face, to the individual's organization, to their interests, skills, etc. Project activities, documents, and ideas can also be added to the hypermedia repository. Because of the multimedia capabilities of HTML and helper applications, the repository can contain a broad spectrum of information.

Design Intent provides a set of project-specific content creation tools. Team members can add, annotate, and supplement the information in the Design Intent repository. Annotations can be textual or can include HTML commands. In addition to annotating existing documents, users can create new documents using HTML editors or multimedia authoring applications on their workstations. The document can then be deposited into the multimedia repository using a helper application that we implemented called "HTTP Upload" (see section on "Uploading Files to be Integrated into the Repository").

Design Intent also provides a set of project-specific communication tools that enable all project members to work collaboratively to construct a consistent understanding of the development effort. Team members are able to use electronic mail, bulletin-board like mechanisms, and annotations to engage in an asynchronous dialog with other team members.

Figure 1: Project Team and Project Member Pages

Finally, Design Intent contains a set of tools for browsing and searching the multi-media repository. Users may request that a notification mechanism inform them about additions, annotations, or pending events.

NYNEX Portholes

Like Xerox Portholes [5], NYNEX Portholes enables individuals to maintain an awareness of their virtual work group. However, our version includes substantial extensions to support the sociality of work. One such extension allows users to select who is in their virtual work group (see Figure 2). Users also have some control over who can access their image. Users belonging to the same groups (i.e., organizational-related and project-related groups) may select each other's images without restriction. However a user can control whether their image can be viewed by users in other groups (see "Groups I Export to" in Figure 2).

Color or grayscale JPEG images (depending on workstation and camera capabilities) of users are captured every 5 minutes via framegrabbing software on a user's computer (Macintosh, Unix, and Windows PCs) and transferred via TCP/IP to an image repository. Users can view a selected set of images in a Web browser and have the browser update the images every 5 minutes (see Figure 3). Users with a large community can use small images to save space. Currently, we have 26 cameras connected to our Portholes network, including four public spaces and two cameras at a remote location (University of Colorado). We are in the process of adding more remote locations. Images and user information are exchanged between the remote locations every 5 minutes.

NYNEX Portholes provides several other features not in Xerox Portholes. Our users are able to control the clarity of their

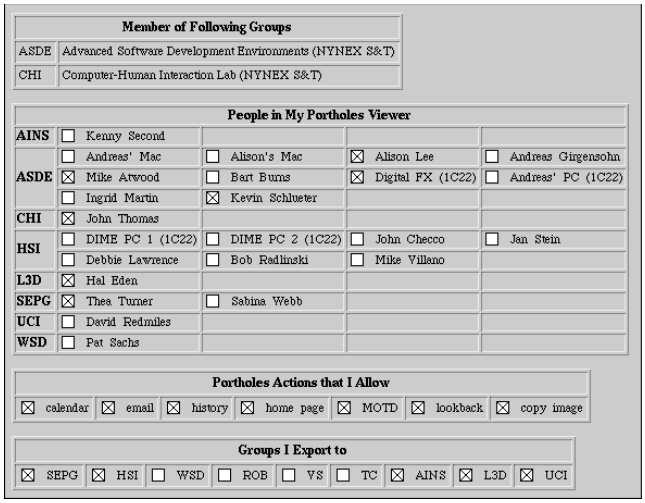

Figure 2: User Preferences for NYNEX Portholes

own image. This allows users desiring more privacy to decrease the amount of detail in their image (and hence information about their current status) [3].

We also add to the notion of using images to create awareness by linking the images to other pertinent information about the user. Each user can select the information that they wish the system to present when someone selects their image

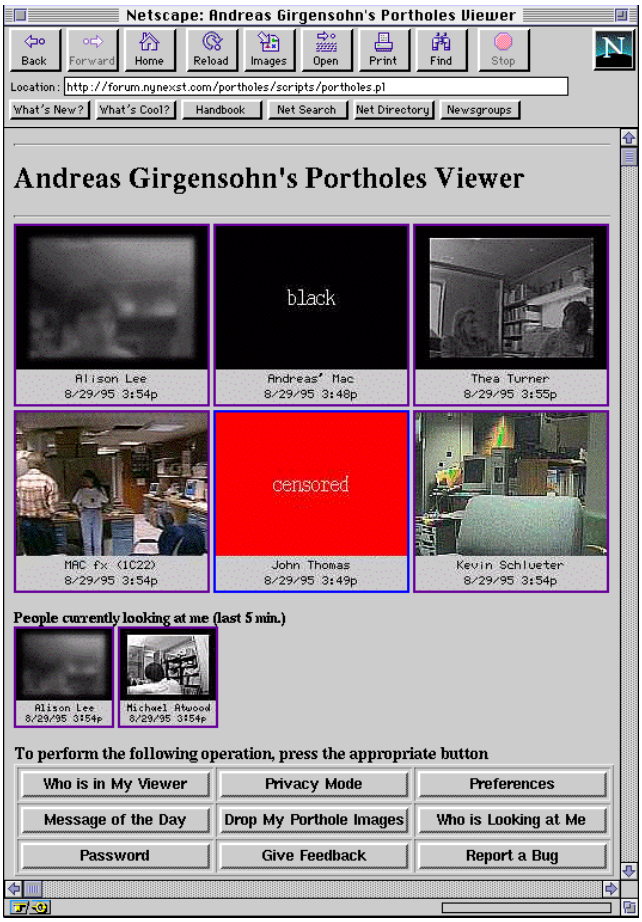

Figure 3: NYNEX Portholes Viewer

(see "Portholes Actions that I Allow" in Figure 2). This includes *formal* information like the group (i.e., physical group and project group) that the person belongs as well as the person's calendar. Users may share *informal* information by allowing other users to access their home page or by allowing other users to see their message of the day. Finally, users may also allow other users to view a crude time-lapse animation made from the last hour of their images to obtain an integrative view of their status (this can be useful to discover, for example, that a co-worker has been on the telephone for much of the past hour and perhaps should be allowed to get work done without another interruption of a phone call).

Because of the time-lapse feature, images are retained for an hour instead of about 5 minutes. Users wanted the ability to remove embarrassing or inappropriate pictures. NYNEX Portholes allows users to inspect their last hour's images and replace any of them by a frame containing the word "censored" (see Figure 4 and John Thomas' image in Figure 3). Users also wanted the ability to see who may be looking at their image. In response, we added a "look-back" feature where small images of those who have recently accessed that user's image appear at the bottom of the viewer (see Figure 3).

Figure 4: Last 12 images for a user

In addition to providing awareness information, NYNEX Portholes allows users to initiate spontaneous informal communications with co-workers. Users can invoke video conferences over the analog network, send electronic mail, and obtain phone and fax numbers. We are implementing a feature where telephone numbers can be automatically dialed. Our intent is to turn NYNEX Portholes into a communications nexus for group members, a function that extends NYNEX Portholes beyond providing pre-communication information to also allow users to initiate communications.

RELATED WORK

There are several hypermedia systems that focus on the support of collaborative work. The Virtual Notebook System (VNS) [18] is a good example. Its WYSIWYG interface for creating documents and its support for concurrency make it much better suited for shared authorship of hypertext documents than the WWW. The Knowledge Management Sys-

tem (KMS) [1] is another example. It provides a single, logical database that is physically distributed across workstations and file servers. Concurrency control and protection mechanisms help to reduce interference among multiple authors and to restrict access to sensitive data. KMS supports communication without using special purpose capabilities like electronic mail or bulletin boards. Specifically, KMS frames, the main information unit, easily function as mailboxes, bulletin boards, and adhoc discussion areas because users may attach and reposition comments in a frame. Finally, KMS provides a general-purpose block-structured programming language that is similar in scope to Hypertalk [9] for extending the functionality of KMS (i.e., creating and manipulating KMS structures). SEPIA is another example of a hypertext system that supports collaborative authorship [12]. In particular, it provides capabilities for supporting both synchronous and asynchronous modes of collaboration and loosely and tightly coupled work.

There are many other Hypertext systems that are better suited than the WWW shell for some aspects of the collaborative applications we have described in this paper. For example, a meeting notes application (see Figure 7) could easily be written in Lotus Notes or in one of the hypertext systems that support the collaborative creation of document (e.g., SEPIA, KMS). However the WWW Shell has some decisive advantages over these systems. The CGI script interface makes it easy for developers to add functionality. It also makes it possible to process the user input in various ways. The project timeline document shown in Figure 5 relies on these features to generate an image on the fly in response to the user input.

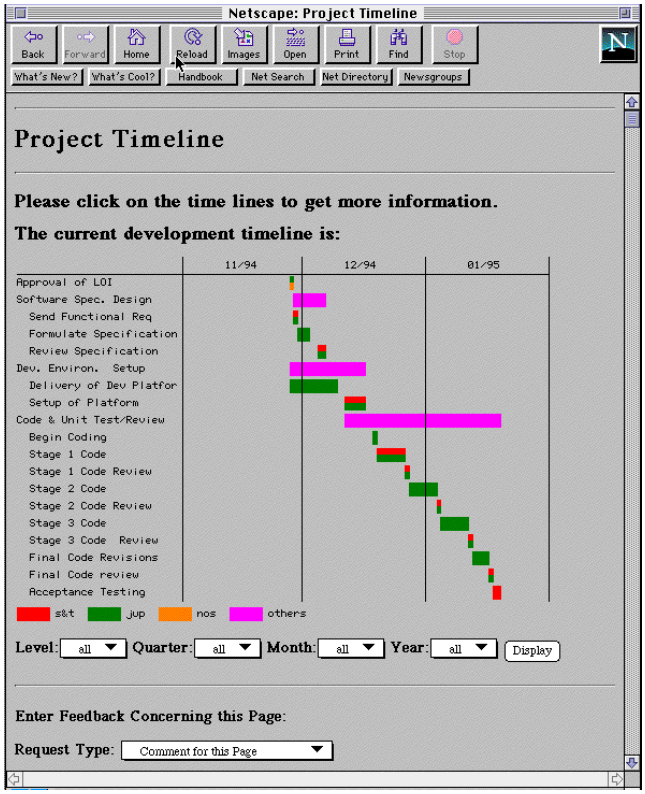

Figure 5: Project Timeline

Another advantage of the WWW Shell is its wide-spread distribution. This has created a large user base that is already familiar with the user interface. The native implementations for different platforms make the distribution of the application easier than for example an application which solely runs under X Windows (the use of which would also incur a performance penalty).

Another reason to prefer the WWW Shell is that most collaborative hypermedia systems concentrate on the shared authorship of information. While shared authorship might address many of the issues in our Design Intent application, it does not address the problems we faced in implementing NYNEX Portholes. Also, while other collaborative hypermedia systems provide more features, the open architecture of the WWW Shell makes it better suited for rapid prototyping of collaborative applications. Perhaps once the needs of an application are better understood, it will be advantageous to reimplement it in a more feature-rich environment.

ADVANTAGES OF USING THE WEB FOR OUR APPLICATIONS

Our approach for exploring the kind of collaborative applications needed to support group communication has been to work with real project teams to identify communication needs and to support those needs using collaborative applications. In this setting, we need a development environment in which we can quickly prototype, test, and evolve the applications so that they can have an impact on the work processes that we are attempting to improve.

While there were other options for development environments, user needs (e.g., support for different user hardware platforms), evolving requirements, and environmental constraints (e.g., users were geographically distributed) were the primary factors in our selection of the World Wide Web Shell as a development environment. In the course of developing the two collaborative applications, Design Intent and NYNEX Portholes, our experience has shown that the WWW Shell is a suitable development environment for the exploratory development of these collaborative applications. In this section, we discuss a number of characteristics of the WWW Shell which facilitated our efforts. These characteristics are not only important for addressing our application requirements but they highlight broader issues that are not collectively addressed by any single CSCW or Hypertext system. We have two goals in this discussion: 1) to point out the “pearls” in the WWW Shell and 2) to point out the value of having a flexible prototyping environment for collaborative applications.

Portability

Our users want to do their work on the platform of their choice and are normally unwilling to learn a new platform or interaction style. Therefore our collaborative applications must function in the X Windows (Unix), Microsoft Windows and Macintosh environments and follow familiar interaction styles. This is in general a difficult requirement that necessitates the use of a cross-platform development system and careful attention to writing portable code. By using the Web, our users need only a browser and set of helper appli-

cations for each platform supported, which are not usually onerous requirements. The interaction style that results is then familiar because our users are familiar with the web and browsers. The non-user-interface application code executes on a single computer (the server) which means this code need only work on one platform.

Rapid Prototyping and Distribution

It is important to respond quickly to user feedbacks by being able to rapidly add new features (or modify old ones) and to quickly distribute the new version to all users. In using the WWW Shell, all application code resides on a server. Using an interpreted language such as Perl lessens the time between making a change and being able to execute the new application. To facilitate rapid changes, we have created an extensive library of reusable Perl functions that perform tasks such as placement of form widgets, layout of tables, processing of form input, and access to database files. The HTML layout mechanisms allow us to quickly specify the appearance of the user interface in a way that is platform independent and that allows it to be rapidly realized and tested. Because application code is downloaded each time the user accesses it, explicit distribution of the new application is unnecessary and the version in our users' possession is as current as we wish. This lets us test changes on our entire user population, instead of just with a few specially designated testers. It also lessens the consequences of making changes that unintentionally break major parts of the application, because such problems can be easily corrected and the changes are available for the user once it is made on the server.

We were able to make use of applications external to the Web (e.g., spell checkers and electronic mail) to quickly prototype some of the Design Intent functionality. For example, we notify Design Intent users via email when other users add comments or new documents and use Unix "cron" to send an email reminder a day before a scheduled event.

User Identification and Authentication

User identification is needed for the creation and access of customized information for each user. NYNEX Portholes users can set many preferences including the people that appear in their viewer, the size of the images, the groups they want to export their images to, and personal information such as their email address. Users can also dynamically change their visibility mode for their Portholes images (i.e., sharp, normal, foggy, gray, black — see Alison Lee's image in Figure 3). All the information about individual users is kept in a database and accessed with the user ID. Because a user ID is only provided to a script if the access to documents is restricted, all scripts that present customized information have to be located in a restricted directory. User customization is also used in the Design Intent application, where users are not shown comments about documents that they do not express interest in. Furthermore, the user IDs enable us to associate authors' names to their contributions.

We also use the HTTP server authentication feature for access control. In NYNEX Portholes, we restrict access to those who have Portholes cameras (and whose images therefore appear on the system). We did this to prevent Portholes from being used as a surveillance system — only users who are willing to provide images may use the NYNEX Portholes viewer. A related use of authentication is to ensure confidentiality of project documents. Our Design Intent users did not want the information they provide to be accessible by anyone who discovers the correct URL. By using authentication, such information is made available only to project members.

When user identification and authentication is used, the administrator of the application must decide how to maintain the database of user names and passwords. If users chose to use their Unix passwords in the collaborative applications, then this password can be extracted from the Unix password file so that the administration of the authentication database can be fully automated.

Information Sharing and Communication Support

A key requirement of our collaborative applications is that they allow a community of users to share information. The WWW provides a transparent framework for users to access a centralized database of information that can be managed and updated by CGI scripts. In the Design Intent application, project information can be published by any user and only the author can change its content. Users have immediate access to the published information, along with any changes. Since the information repository resides on the server, the authors do not need to be concerned that other users will access obsolete information.

Users can critique, clarify or comment on published documents via an annotation mechanism that is common to all Design Intent displays. Annotations are made via the forms facility and are dynamically folded into the appropriate document and immediately presented to the user by CGI scripts (see Figure 6). The resulting discussion thread, created through the series of annotations, resembles the discussions that occur on a *bulletin board*, *mailing list* or *newsgroup* service. Users interested in the discussion may be notified via electronic mail when new comments are made. Email is sent using Unix mail when the annotations are processed by the CGI scripts. Aside from the annotation mechanism as a form of communication support, users can also send electronic mail by clicking on a "mailto:" link. We are currently integrating other means of communication (i.e., video conferencing and telephony) into NYNEX Portholes using CGI scripts. Much like the "mailto:" feature for electronic mail, users will be able to click on items like a phone number, for example, to initiate a phone conversation.

Uploading Files to be Integrated into the Repository

We developed forms and CGI scripts to allow users to add to or modify information in the Design Intent Repository. Similar mechanisms are used to allow users to change their personal information in NYNEX Portholes. While this allows users to add textual as well as HTML information, users also want to add non-textual information such as sounds, images and QuickTime or MPEG videos. Unfortunately, such things cannot be entered via the HTML form facility making it difficult for users to contribute them (see the "Interaction

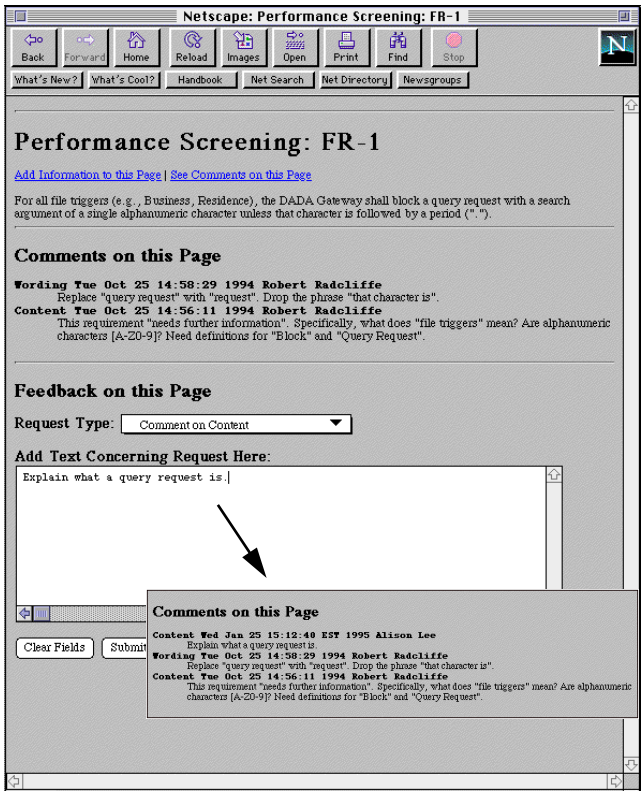

Figure 6: Annotations to Project Requirements

Asymmetry" section under "Limitations"). Furthermore, not all Design Intent users want to use the form mechanism to prepare documents. They would rather create documents using their favorite application software (e.g. MS Word) and then transparently submit them.

To solve the problem of transferring files in a user-friendly manner from a client machine to the server, we developed the "HTTP Upload" helper application (currently only implemented on the Macintosh). Figure 7 illustrates a user requesting that a file be attached to a meeting note which is being created. When the user clicks on the submit button (after having chosen "Upload a File Only" from a menu), the server script sends a specific MIME type to the web browser which in turn triggers the "HTTP Upload" application. The helper application prompts the user (using a standard file dialog) for the file to be uploaded. The upload application translates the Macintosh file type into a MIME type so that the script can pick an appropriate file name extension before attaching the document. The helper application then contacts the HTTP server and later communicates with the Web browser via an AppleEvent. The server passes two URLs along to the upload application. The first points to a server script that will place the uploaded document in the part of the repository where meeting notes are kept. The second URL is sent to the Web browser after successful completion of the upload.

This roundabout method is necessary because current Web browsers do not support document uploading even though the HTTP server has no problems with it. Since we wrote the

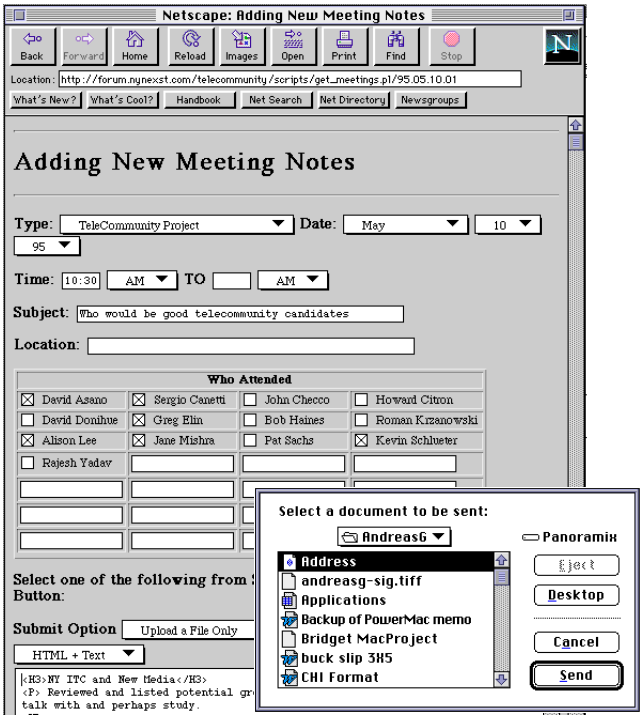

Figure 7: HTTP Upload

first version of this paper, version 2.0 of the Netscape Navigator has added a file upload facility to HTML forms. Interestingly, the request for comments which provides the basis for this new feature [13] describes a helper application to be used as an interim solution that is very similar to our "HTTP Upload" application. This shows that our need has been recognized as an important feature by the Web community.

Interactive Presentations

Users often want to do more than simply view static presentations of information. They want to control the way it is presented and add their own knowledge and opinions. For example, in the project timeline document (a component of the Design Intent application), users can inspect a timeline image and request more or less information to be shown in the timeline by specifying the desired time interval and amount of detail (see Figure 5). Users can also add their comments to the timeline with our annotation facility. All this is possible because the WWW Shell allows us to dynamically generate the image from the project data and the user selections via CGI scripts. The shell also provides a facility for passing the location of mouse clicks in an image to the server, where it can be processed by CGI scripts. We use this to intercept mouse clicks on the timeline image and display information associated with the particular timeline element that was clicked on.

Automatic Updates

Early on, we were unable to have automatic, or non-user-initiated updates. In NYNEX Portholes, a user's viewer needs to be updated automatically at regular intervals to stay current. Initially, users needed to press the "Reload" button pe-

riodically. This is undesirable because Portholes provides background awareness information (roughly peripheral information that we do not focus our conscious attention on) about one's co-workers. With Netscape's introduction of the client-pull feature, we are able to include information in the web page specifying when the client's next access of the document should be made without user intervention. This provides enough control to synchronize the client to the regular server update of the images so that the users can see them shortly after they are generated.

LIMITATIONS

Computer-mediated collaboration ideally provides both resources for information sharing and support for communication. Early applications developed with the WWW Shell focussed on the Web as resource for sharing information. Increasingly, researchers and developers have begun to use the Web as a medium for communication. There have been enhancements to the WWW Shell to facilitate both information sharing and communication, however several limitations remain. It can be difficult to use the WWW Shell for developing some aspects of collaborative applications, especially those that require more sophisticated user interfaces, better interactions amongst users and/or systems, and complex architectures and structures for representing information. This section discusses specific limitations in more detail and closes by discussing how some of these limitations are being addressed in new web browsers.

Lack of Immediate Application Dependent Feedback

User interfaces can sometimes be improved by providing immediate application-dependent feedback. Examples include rotating objects or using rubber band lines in two-dimensional draw programs as the user drags the mouse. The former requires access to a great deal of application state and is probably not possible in Web applications because of the current heterogeneity of Web network links — some links are simply too slow to permit the transmission of a great deal of application state quickly enough to provide usable feedback. However, if widgets can provide simple feedback like the rubber band example, then no network traffic would be generated because there is no need to access the application state. Having this kind of simple immediate application-dependent feedback would greatly improve Web interfaces. One possible improvement would be highlighting the selectable areas in graphical images when the user moves the mouse pointer into them. Often a single graphical image is used for a stack of buttons and it is sometimes unclear which button the mouse tracker is in. The ability to maintain image maps on the client side, as introduced in Netscape 2.0, partially addresses this issue.

Form Submission Only When Submit Button is Pressed

In HTML, form information is only sent to the server after the user presses the submit button. If menu choices and other changes to form widgets could cause submits, we could implement forms where the widgets change (and appear and disappear) depending on context. Currently, we are trying to use HTML to implement a questionnaire about people's communications. Usability of the questionnaire would be greatly improved if it was possible to ask different information of the users depending on the type of communications contacts provided by the user (e.g., users would be asked about the durations of meetings and the lengths of faxes, but not vice versa). Presently this is not possible because the server cannot change the form immediately after the user has selected the contact type from the contact menu. It must wait until the submit button is pressed. If a submit could be triggered by a change in the contact menu, this problem would not exist. Having submits triggered by menu or radio button selections would also allow better interfaces for Web pages that display one of several variants of a section (e.g., one of several views of a weather map) as the variant displayed would change immediately upon menu or radio button selection. Having submits triggered by keystrokes (perhaps the space bar or carriage return as triggering on every single character would incur excessive network overhead) would allow interactive spell checkers, editors with automatic command completion, and limited on the fly text formatting.

Server Initiated Interactions

Only the client can initiate an interaction with the server. This causes problems if the client needs to view information that changes from time to time. It would be best if the client could register with the server and then receive updates if the information it is displaying changes. Unfortunately, the HTTP model does not support server-initiated document updates.

In version 1.1 of Netscape, periodic polls by the client are supported and can get around some of the situations where server-initiated updates are desired. However, it would be preferable if polling would be less costly so that the client could poll in relatively short intervals (maybe once every 30 seconds compared to the five minute Portholes interval). Polling is costly because the entire contents of the URL are fetched on each poll and redisplayed even though none or just a small part may have changed. In our Portholes application, this means each poll causes the fetch and redisplay of all images even when only a few have changed. It would be preferable if the polling mechanism would only retrieve, or at least only redisplay, changed information.

The "server-push" feature of Netscape maintains a connection between the server and the client so that the shortcomings described above are overcome. NYNEX Portholes uses this for animating the images of the past hour. Such a permanent connection would also overcome some of the problems of statelessness discussed in the next section. Unfortunately, this connection imposes a heavy burden both on the client and the server. Also, because each inline image requires its own connection, the NYNEX Portholes viewer could not be implemented in this fashion because it would require too many parallel connections. What we really need is a lightweight logical connection maybe implemented through the use of UDP packets.

Statelessness

Interactions between client and server are stateless. This means that each request by a client will find the server in exactly the same state. Some applications require multi-step in-

teractions which need to maintain different states between interactions. In the HTTP model, modal interactions are only possible if the client keeps track of the state (as part of the current document) and passes it to the server with every request. A model has been proposed whereby the updated state is returned as part of the document returned by the server to the client. This model can become very cumbersome as has already been reported elsewhere [4]. Some Web applications store the state on the server and use "magic cookies" to refer back to it but this would overcome only some of the problem and a better mechanism would be desirable.

Layout

HTML provides limited control over the layout of a document. This prevents the designer from arranging a form in a way suitable for an individual application. Dynamic layout changes that are dependent upon user interactions can not be done in a fashion similar to Dynamic Forms [8] which would make forms much more user-friendly. Changes are only possible after the form is submitted by pressing a button and these changes require the redisplay of the whole form. Widgets can only be placed in a text flow and geometric constraints provided by user interface toolkits can only be used to a very limited degree. HTML tables can be used to work around some of these limitations. Even the layout of text is much less flexible than standard word processors.

Missing Widgets

The standard widgets for continuous numeric input — sliders and dials — are missing from the HTML forms facility. Even if the forms widget set was expanded, it is likely that many useful widgets would be missing, making it desirable to have some platform-independent way to specify custom widgets. One possibility would be to use a technique similar to the one used in the NeWS [11] window system. In NeWS, an interpreted language with a comprehensive imaging model (Postscript) was augmented with an event handling mechanism to permit the implementation of user interface widgets. The interpreted code could be downloaded and executed on the window server regardless of platform. Adding the ability to define forms widgets analogously (perhaps using Java [10] as the interpreted language) would eliminate the need to provide a large selection of widgets in future versions of HTML forms. It would also solve the problems of interactivity and the lack of immediate application dependent feedback.

Multiple Windows

Web applications cannot open new windows to partition their output. Developers can tile a window by using layout techniques to simulate a fixed number of windows. This technique does not work when an arbitrary number of windows (usually dependent on user actions) is desired. In the questionnaire being implemented in HTML, it would be desirable to have a separate window appear every time the respondent wants to record a new communications contact. This interface seems better than displaying only the last *n* communications contacts because a respondent is less likely to enter a communications contact twice (which will cause some problems when the data is analyzed).

Viewing Foreign Document Formats

Our "HTTP Upload" application allows users to transfer any type of file to the server, but in doing so can create a "Tower of Babel" situation in which the files uploaded can not be viewed by many users. For example, a Macintosh user did not realize that an uploaded MS Word file could not be viewed by UNIX users. There are two general solutions to this problem. The server can use a translator to convert the document format to HTML format if such a translator exists. Note, this approach generally produces an impoverished version of the document because of the limited markup capabilities offered by HTML or the limited translation provided by the translator. Alternatively, we are considering the use of Adobe Acrobat to convert the documents to PDF format and using Acrobat Reader as a helper application to view the PDF file. This solution preserves the formatting of the original document.

Interaction Asymmetry

Web interactions are somewhat asymmetric. Users can download and view a rich set of media types but are restricted by the forms facility to responses where they pick one of several alternatives or enter text. Some of this asymmetry can be remedied by using clever application-specific textual description languages such as HTML for hypertext entry. However our experience has been that many users are unwilling to learn these languages. Our users want to use their favorite applications to create documents to be processed by our collaborative applications. For example, one of the Design Intent users preferred to use Microsoft Word to create meeting notes. Other people would like to upload images and Quicktime or MPEG videos without having to learn about FTP, about our server directory structure and about HTML.

One way to solve these problems is to have a designated "Web Meister" who is sophisticated technically to convert documents into a usable form and then link them into our web applications. We feel that reliance on such a person weakens the collaborative aspect of the applications and introduces an unacceptable lag between document creation and document availability via the web (this is especially true if the "Web Meister" is only moonlighting in the position as is often the case). We are presently trying to solve this problem using our "HTTP Upload" facility, which is explained in the "Uploading Files to be Integrated into the Repository" section.

Opportunities for Overcoming the Limitations

A reader might think at this point that we are unlikely to be able to convince the appropriate parties to make the browser changes we require. In response we point out that new browsers are beginning to address some of the limitations we pointed out earlier: Named windows can be opened when following links and frames can be used to support the division of a window into subwindows to better structure the presented information. A widget for uploading files has become part of the HTTP forms facility in one browser. We hope that our work and that of others using the Web for col-

laborative application development will continue to lead to the addition of important features in Web browsers.

New browsers are also offering *programmability* and *extensibility*, allowing developers such as ourselves to add the features we require, at the cost of using a more complex procedural/object oriented programming language in place of the simpler, declarative HTML. Because one of our goals is rapid prototyping, the costs and benefits of this trade-off will have to be studied.

The HotJava [10] and Netscape browsers are *programmable* in that they can download small programs, called "applets" written in Java (a programming language resembling C++) and then execute them. Java applets can have richer user interfaces that overcome most of the limitations of using HTML Forms. Netscape provides another simpler language called "LiveScript" that adds limited programmability to HTML Forms and provides a way to integrate forms widgets and Java applets which may facilitate rapid prototyping. LiveScript is still very much a work in progress and thus it is hard to make predictions about its capabilities (and thus about whether we can use it to overcome some of the limitations we encountered).

HotJava is also *extensible* in that it can also be dynamically and transparently augmented by downloading protocol handlers and content handlers that are written in Java. This is a powerful facility that would allow us to overcome many of the limitations we discussed by making modifications to the forms facility through modifying the HTML content handler (in effect we would be making our own version of HTML). Specifically, we could add new widgets to the forms facility, and have buttons and menus to trigger submits. We could also add application-dependent feedback through modifications to the forms facility. However, because other browsers are not extensible in the same manner, we may end up restricting our pool of potential users, which is undesirable in our environment.

CONCLUSIONS

While alternative approaches exist (e.g., using KMS or SEPIA) for developing the collaborative hypermedia applications, the constraints related to exploratory development of the application, support for multiple platforms, user familiarity with WWW, etc. were the prime determinants for using the WWW Shell for our development efforts.The synergy of WWW components offers significant advantages in these areas. It can thus be used for applications like Design Intent, which are somewhat similar to those done with other hypermedia environments as well as for substantially different applications like NYNEX Portholes, which are more typical of those done with CSCW toolkits. It should be pointed out that efforts within the CSCW community to develop toolkits for building collaborative applications [7, 15, 16, 17, 20] have, unlike the WWW Shell, been restricted in scope and in their support for different computing platforms.

The WWW shell does have limitations which restrict the usability and functionality of collaborative applications. We find it encouraging that some of these are being addressed in new web browsers which, much like the collaborative applications we have implemented, are evolving based on user needs. There is reason to believe therefore that the list of the WWW Shell's advantages will grow and that the list of its limitations will shrink. This paper is an effort to encourage others to consider it for exploratory development of collaborative applications and to provoke discussions of its limitations so that an effective consensus can be reached as to how it can be extended and improved.

REFERENCES

1. R.M. Akscyn, D.L. McCracken, and E.A. Yoder. *KMS: A Distributed Hypermedia System for Managing Knowledge in Organizations.* Communications of the ACM, 1988. **31**(7): pp. 820-835.

2. M.E. Atwood, B. Burns, D. Gairing, A. Girgensohn, A. Lee, T. Turner, S. Alteras-Webb, and B. Zimmermann. *Facilitating Communication in Software Development*. In Symposium on Designing Interactive Systems. 1995. New York: ACM. pp. 65-73.

3. V. Bellotti and A. Sellen. *Design for Privacy in Ubiquitous Computing Environments*. In Proceedings of the Third European Conference on Computer-Supported Cooperative Work (Milan, Italy). 1993. London: Kluwer Academic Publishers. pp. 61-76.

4. P. Burchard. *W3Kit, an Object-Oriented Toolkit for Interactive WWW Applications.* 1994. The Geometry Center: Online Technical Report http://www.geom.umn.edu/software/w3kit/.

5. P. Dourish and S. Bly. *Supporting Awareness in a Distributed Work Group*. In Human Factors in Computing Systems, CHI'92 Conference Proceedings (Monterey, CA). 1992. New York: ACM. pp. 541-547.

6. R.S. Fish, R.E. Kraut, and B.L. Chalfonte. *The Video Window System in Informal Communication*. In Proceedings of CSCW'90, Conference on Computer-Supported Cooperative Work. 1990. New York: ACM. pp. 1-11.

7. S.J. Gibbs. *LIZA: An Extensible Groupware Toolkit*. In Human Factors in Computing Systems, CHI'89 Conference Proceedings (Austin, TX). 1989. New York: ACM. pp. 29-35.

8. A. Girgensohn, B. Zimmermann, A. Lee, B. Burns, and M.E. Atwood. *Dynamic Forms: An Enhanced Interaction Abstraction Based on Forms*. In Proceedings of INTERACT'95: Fifth IFIP Conference on Human-Computer Interaction. 1995. London: Chapman & Hall. pp. 362-367.

9. D. Goodman. *The Complete HyperCard Handbook*. 1987. New York: Bantam Books.

10. J. Gosling and H. McGilton, *The Java Language Environment: A Whitepaper*. 1995, http://java.sun.com/whitePaper/javawhitepaper_1.html.

11. J. Gosling, D.S.H. Rosenthal, and M.J. Arden. *The NeWS Book: an Introduction to the Network/extensible Window System*. 1989. New York: Springer-Verlag.

12. J.M. Haake and B. Wilson. *Supporting Collaborative Writing of Hyperdocuments in SEPIA*. In Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW'92). 1992. New York: ACM. pp. 138-146.

13. E. Nebel and L. Masinter. *Form-based File Upload in HTML*. 1995. Network Working Group: Request for Comment 1867, ftp://ds.internic.net/rfc/rfc1867.txt.

14. R.W. Root. *Design of a Multi-Media System for Social Browsing*. In Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW'88). 1988. New York: ACM. pp. 25-38.

15. M. Roseman and S. Greenberg. *GroupKit: A Groupware Toolkit for Building Real-Time Conferencing Applications*. In Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW'92). 1992. New York: ACM. pp. 43-49.

16. M. Roseman and S. Greenberg. *Building Real Time Groupware with GroupKit, A Groupware Toolkit.* ACM Transactions on Computer Human Interaction, 1995 (in press).

17. T.D. Rüdebusch. *Development and Runtime Support for Collaborative Applications*, in *Proceedings of the Fourth International Conference on Human-Computer Interaction (Stuttgart, Germany)*, H.-J. Bullinger, Editor. 1991. Elsevier Science: Amsterdam. pp. 1128-1132.

18. F. Shipman, R. Chaney, and T. Gorry. *Distributed Hypertext for Collaborative Research: The Virtual Notebook System*. In Proceedings of Hypertext'89 (Pittsburgh, PA). 1989. New York: ACM. pp. 129-135.

19. J. Short, E. Williams, and B. Christie. *The Social Psychology of Telecommunications*. 1976. New York: John Wiley & Sons, Ltd.

20. T. Urnes and R. Nejabi. *Tools for Implementing Groupware: A Survey and Evaluation.* 1994. York University, Toronto: Technical Report CS-94-03.