MetaWeb: Bringing synchronous groupware to the World Wide Web

Jonathan Trevor, Thomas Koch and Gerd Woetzel
German National Research Centre for Computer Science (GMD FIT.CSCW)

Abstract The World Wide Web is increasingly seen as an attractive technology for the deployment and evaluation of groupware However the underlying architecture of the Web is inherently stateless - best supporting asynchronous types of cooperation This paper presents a toolkit for application developers, MetaWeb, which augments the Web with basic features which provide new and legacy applications with better support for synchronous cooperation. Using three simple abstractions, User, Location and Session, MetaWeb allows applications to be coupled as tightly or as loosely to the Web as desired The paper presents two distinct applications of MetaWeb, including the extension of an existing application, the BSCW shared workspace system, from which a number of observations are drawn

1. Introduction

The World Wide Web (Berners-Lee et al., 1994) merges the concepts of hypertext and networked information to provide an easy but powerful global information system based on two public and simple standards: the HyperText Transfer Protocol (HTTP) and the HyperText Markup Language (HTML). The Web consists of many independent servers, which accept and service requests for information in the HTTP protocol from clients (Web browsers), such as Netscape.

This simple architecture has now become established as an important platform for developing, deploying and evaluating CSCW applications. Almost any new CSCW system has some type of Web interface, whether it is purpose built for the Web, such as BSCW (Bentley et al. 1997), or provides generic access to features in non-Web applications, as with POLIWEB (Freund, 1996) Even existing systems like LOTUS NOTES with DOMINO have been extended to present their interfaces as Web pages.

There are many reasons for the success of the Web, and its attractiveness for developers of CSCW applications. Dix (1997) provides an analysis of the advantages

J. Hughes et al. (eds.), Proceedings of the Fifth European Conference on Computer Supported Cooperative Work, 65-80.

and problems of the Web for cooperative systems, highlighting a number of reasons behind its success: a core initial user community, the integration of existing information, the use of de facto standards, the spanning of organisational boundaries, and a software platform which is public domain, cross-platform and extensible. Perhaps the most important reason presented by Dix is that the Web has reached a critical mass, the point where the benefits outweigh the costs for the user - a traditional problem for CSCW systems (Grudin, 1988). These advantages have made the Web an important platform for CSCW development.

2. Limitations for Groupware on the World Wide Web

The basic client-server architecture of the Web (Figure 1) is sufficient for serving primarily static pages of information from fixed locations in the remote file system. However for applications which want to present information differently to different users, or where the information available changes frequently, the basic server needs to be extended. The most common method of doing this is through the Common Gateway Interface (CGI) (Figure 2). This interface describes how HTTP requests can be delegated by the server to some server-side application. The delegated request can then be processed by the application before returning a response (typically an HTML page) to the server, which returns it to the client.

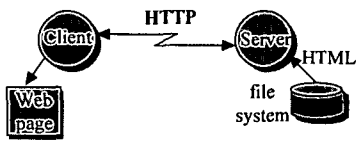

Figure 1: The WWW architecture

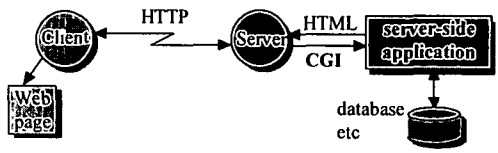

Figure 2 Integrating applications in the WWW

The problem for CSCW applications which adopt the Web as a platform for application development is that they inherit a key property of the Web - that the protocol it uses (HTTP) is inherently stateless. No information is stored between requests, and therefore the server (or application) has no idea what page the client is now browsing, how old that page is, and so on. The user could request a page from an entirely different server or application and the previous Web application would remain unaware of this change. The problem is bi-directional as the client invariably sees just a "snapshot" of the application's state. If something changes in the application, there is no way for the user to know this until he reloads the page (issues another HTTP request to the application).

This problem restricts those types of cooperative system supported by the Web to the "different time" half of the 2x2 classification of CSCW systems described by Ellis et al. (1991) - those supporting asynchronous local and distributed interaction. As such, cooperative tools and applications which require any significant degree of synchronous interaction between the users, and/or the application, such as chatting or shared whiteboards, cannot be supported by the current Web architecture.

A possible solution is to *modify the HTTP protocol* to incorporate more state information, to allow a stronger bi-directional coupling of the client to the server. Indeed, the NETSCAPE NAVIGATOR browser has introduced minor extensions to the HTTP protocol to satisfy some of these requirements (using "cookies" to maintain state and "Javascript" to move some computation from the server to the browser). An alternative approach to the problem, and often a consequence of the first, is to *modify the WWW client* to include specialised synchronous support. This may be building a complete replacement or embedding new functionality in an existing browser, using the Common Client Interface (CCI) or Java for example.

Unfortunately, in the context of the Web, these types of changes impact directly on the critical mass problem. For example, a specialised client means that effort required to use the Web application is drastically increased as a special piece of software must be located, downloaded and installed in addition a standard Web browser. *Any solution which affects the advantages the Web brings, such as changing the de facto standards or requiring a specialised software platform needs to be avoided* (Trevor et al., 1996). Therefore, we require some means of supporting synchronous groupware which does not change the way the Web currently works.

A solution to this dilemma is to provide some *additional* infrastructure which supplements the Web-based application with an architecture which allows information to flow both ways between the user and the application, supporting immediate feedback of the user's actions to the application and notifying changes of application state to the user. This additional infrastructure should not require any special client software or affect the standard protocols.

3. Addressing the shortcoming: Existing systems

A number of existing systems support applications requiring additional synchronous functionality, and may provide the additional infrastructure mentioned above. We separate these into general and Web-specific systems.

3.1. General support for synchronous collaboration

There are several non-Web oriented systems and toolkits which focus on addressing the general needs of synchronous CSCW application developers. These toolkits and systems include MMCONF (Crowley et al., 1990), RENDEZVOUS (Patterson et al., 1990), and GROUPKIT (Roseman and Greenberg, 1992). A number of concepts for synchronous collaboration development are common to these systems:

- **Session management;** Ellis et al. (1991) define a ***cooperative session*** as:

  > "a period of synchronous interaction supported by a groupware system Examples include formal meetings and informal work group discussions." (p.46)

  Most toolkits provide some mechanisms for applications to model and manage cooperative interaction between users, often grouping them into activities, rooms, conferences etc.

- **Mechanisms for accessing shared information**; mechanisms may be toolkit specific, such as who are the users in a session, or may be application-specific (provided by the application itself) to be shared between different instances of the application.
- **Support for application-specific message exchange;** Cooperative applications are often distributed across networks and require support for sending messages to each other.
- **Access control;** Necessary to control and restrict access to information and functionality available in the toolkit and application.

Unfortunately there is little support for the development or integration of these concepts into the Web itself. The emergence of the Web as a viable architecture for groupware, combined with its weaknesses, means that these existing (pre-Web) toolkits poorly address many new problems, such as how can the provided functionality be integrated with a Web page? How does the notion of a Web page relate to the concept of a cooperative session? and so on. In addition, the previously "desirable" quality of being platform independent is no longer merely desirable but critical within the context of the Web. Therefore, while the concepts are important for supporting synchronous applications, there is no simple or easy way to support Web-based applications in these toolkits.

3.2. Web-specific support for synchronous collaboration

The need for synchronous cooperation support on the Web is also addressed by a number of Web-oriented applications and toolkits, which attempt to couple their features more closely with the existing Web architecture and concepts. Some of the most notable of which include: YARNWEB (Woo and Rees, 1994) which supports chatting between users; VIRTUALPLACES (Ubique, 1996) where Web users can see and meet each other, GROUPWEB (Greenberg and Roseman, 1996) which supports collaborative navigation and allows groups to share pages in a relaxed WYSIWIS view; GROCO (Walther, 1996) which allows the development of Electronic Meeting Systems; and COWEB (Jacobs et al., 1996), which provides cooperative form-filling by dynamically transforming existing pages.

Unfortunately, there are problems with each of these toolkits. The most important is *restrictive cooperative session coupling to Web pages*. For example, in VIRTUALPLACES and GROCO a group always interacts within the boundaries of a single Web page, while YARNWEB only uses the Web as a launching point to a completely separate synchronous communication service. GROUPWEB and COWEB allow interaction over several pages in a group, but provide *no awareness of other users beyond the current page* - users must somehow navigate to the same page before they become aware of each other and interact. These limitations render them incapable of supporting applications whose sessions may span several pages, or conversely sessions where a single page acts as an access point to an application which supports distinct groups of users. Different applications require different ways of mapping their cooperation onto Web pages and cooperative session support on the

Web should support variations ranging from one session per page, to one session for a set of pages, to different sessions on the same page, to no coupling at all, where sessions exist without a corresponding page.

Another significant drawback of most of these systems is their *platform- and browser-dependence*. YARNWEB works with all browsers, but is restricted to the X-WINDOWS system. VIRTUALPLACES is a platform- and browser-dependant helper application. GROUPWEB is based on a specialised browser. GROCO and COWEB both rely on functionality only available in a now obsolete alpha release of Java and the HOTJAVA browser.

Finally, all these systems (except GROCO) provide *very specific solutions to specific problems,* or types of synchronous interaction, making them unsuitable as general toolkits to build and support CSCW applications on the Web. Although GROCO has a number of other problems, it demonstrates that through the use of mobile code (Java) and the provision of a generic infrastructure for synchronous groupware, the Web architecture *can* be extended to more fully support CSCW.

3.3. Goals for supporting Web-based synchronous applications

Section 1 highlighted reasons behind the success of the Web as an infrastructure for supporting CSCW applications, one of the most important being critical mass. Two aspects of this are the use of standard protocols and platform independence. Section 2 showed how the Web's support for applications requiring more synchronous interaction is flawed and argued that any solution should not affect the aspects which make the Web so successful. This Section has presented a number of existing systems, both general and Web-specific and has highlighted shortcomings in both.

From these studies of existing systems and their problems we argue that any system which attempts to support synchronous cooperative work on the Web should adopt the concepts commonly supported by general toolkits (session management, access control, accessing shared information and support for application specific message exchange) while addressing the additional needs of Web-based applications:

- variable coupling of application models of cooperation to the Web,
- platform and browser independence,
- generic support for a range of applications.

The remainder of this paper presents *MetaWeb,* a general architecture and toolkit for developers of synchronous Web-based cooperative applications which provides the common concepts of existing toolkits while accommodating the additional requirements of the Web itself.

4. The MetaWeb Toolkit: Model and implementation

The MetaWeb toolkit provides basic support for the flexible coupling of Web pages to synchronous activities using mechanisms to manage additional "meta" information

about pages on the Web. This includes who is browsing the pages and which cooperative sessions are currently running in or around the pages.

MetaWeb provides three abstractions for supporting the cooperation of Web-based applications: user, location and session. A **user** is the representation in the application of the person interacting with the toolkit. Users *visit* **locations** which are places where group interaction can take place. A location may be a virtual representation of a physical room or correspond to a Web page. A location *hosts* zero or more **sessions.** Sessions are a medium through which communication can take place between users. A session may *contain* zero or more users (its membership), and may place restrictions on that membership. Figure 3 shows these concepts and their relationships.

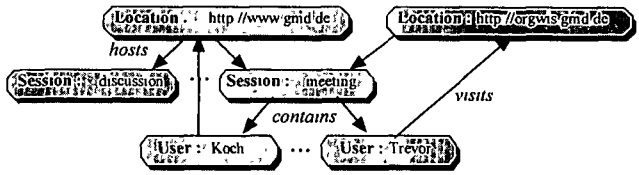

Figure 3. Example of Users, locations and sessions in MetaWeb

MetaWeb places no restrictions on how the notions of location and session are mapped to Web pages. Indeed, the concepts can be tightly coupled to the Web by assigning one location per page (as in Figure 3), and one session per location. In such a relationship, as a user loads different pages he "moves" between different locations in MetaWeb and on each page may see the users at that location/page. Different mappings between these concepts and the Web allow the various associations of the application's cooperative sessions to the Web listed in Section 3.2. In Figure 3 the "meeting" session is hosted by two locations (and Web pages in this mapping) while the "discussion" session can only be found in one. Indeed, Figure 3 shows users at different locations ("trevor" and "koch") in a common session ("meeting").

Although these relationships do not support the nesting capabilities of other session models, we believe that this simplified model *does* sufficiently support many Web-based cooperative applications in a straightforward and understandable manner. A much more complex model which may support every case is bound to be more difficult to understand, implement and use.

4.1. Object model

Each of the three concepts has a corresponding object type in MetaWeb. Applications make use of these concepts by creating instances of the object in MetaWeb. The MetaWeb toolkit maintains these objects and their relationships, which may be available to any application using the toolkit.

The user object is a representation of an application instance connected to the MetaWeb system. This application may present an interface to a real person, or may be an agent. The object is created when the application connects to the MetaWeb

server (see below) and is used to identify the user in MetaWeb. Each user object has two default properties, a name, and a reference to its location.

Session objects maintain the current members of a session (as a list of user object references), an application-defined description of the session, and a list of potential members of the session (defined by the creator of a session). All remaining session behaviour and functionality, such as invitation, history maintenance and floor control, are responsibilities of the application itself.

Location objects have an associated list of zero or more session identifiers occurring at that location. This list may be changed to allow sessions to be attached or removed during the location's lifetime. Each location object also has a text identifier, which can be referenced to uniquely identify that location object in MetaWeb. For example, this identifier could be the URL of the page the location represents (as in Figure 3), or the number of a room.

In addition to the object-specific methods and values, instances of these objects can have any number of additional name-value pair attributes. This allows applications to add or remove application-specific named values from the objects during runtime. For example, an application may want to provide new user attributes, such as the user's phone number, or to add a description to its sessions.

4.2. Architecture and API

Figure 4 shows how MetaWeb extends the Web architecture through an additional client/server-system using its own protocol. This architecture can be split into four separate areas: application, API, client and server.

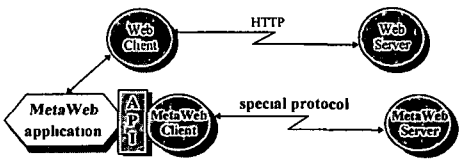

Figure 4. The MetaWeb architecture

The MetaWeb application is a part, or all, of the cooperative application that the developer builds. It is the developer's responsibility to present an interface to the user, take information from MetaWeb, display it in a suitable manner, translate user input into calls to MetaWeb and so on. This is where the application-specific semantics are supported and MetaWeb places no restrictions about what these are.

The application interacts with the MetaWeb toolkit through an API. This is a set of classes and methods which manipulate the object model (users, locations and sessions) and hide much of the underlying complexity from the developer.

The MetaWeb client provides two main services. Firstly, it acts as a representation of the user in MetaWeb. Whenever another application or the MetaWeb server needs to communicate with a certain user it is this client they refer to. Secondly, the client maintains local proxy representations of MetaWeb objects which are stored in the server. Whenever an application wants to use a particular object it is fetched locally,

while a master copy of the object remains at the server. Changes to an object's state are first transmitted to the master, and then to any proxy representations.

The server itself is the heart of the toolkit, maintaining all the objects, enforcing access control, allowing interest registration and the propagation of events (see below). Any number of MetaWeb clients can be connected to the server at any time, supporting interaction across single or multiple applications.

4.3. Communication

Communication is based on the notion of events, which provide an abstraction of state changes in distributed applications. MetaWeb applications create their own events and send them to sessions, specific users or to the system in general (for anyone interested). Events are first created at the MetaWeb client, which sends them to the server, which propagates them to "interested" MetaWeb clients.

"Interest" may either be implicit or explicit. Implicit interest registration is derived from the concepts, where session members automatically subscribe to all events which are sent to the session. Explicit interest registration allows applications to specify various interest contexts (or templates), which describe the set of events an application wants to receive. Events that are sent to the member "anyone" are only received by applications which have registered an interest matching that event. The developer's application itself receives events from the MetaWeb client by registering callbacks with the API, and should define appropriate event handlers for them.

Events are defined as objects with four attributes: the originator (who sent the event), the target (where the event is going), a type (an application-defined text tag to identify the type of event), and a body containing the contents of the event itself. The toolkit defines several types of system event which are used to transport information about state changes in its objects, such as users joining or leaving sessions. Applications are left to define their own generic events which may be used to transport application-specific state changes or information, such as a line of text typed into a chat window.

4.4. Access control

Access control is implemented to restrict access to sessions or private information transported by events. Access to event information can be restricted by the sender of the event and access to sessions (including the events that are sent to sessions) can be restricted by the creator of each session. In both cases access control is specified through one or more user object templates. These limit access to the protected object to only those user objects which have a superset of the name-value pairs in the template. Access control is applied at the server, where access is granted if the user object matches one of the specified templates.

Additional attributes of a MetaWeb object (Section 4.1) can be specified as "hidden". This means that they can only be "seen" by the MetaWeb server. When the object is manipulated by a client these attributes are not available. These hidden attributes can be used in conjunction with access templates to provide authentication.

For example, a session may specify a list of user templates where each template has a single "authenticate" attribute (possibly encrypted using the UNIX "crypt" function). This attribute value contains the username and password of a user. If a MetaWeb client wants to access the session, it must provide a user object with a matching authentication attribute.

4.5. Implementation

The MetaWeb API, client and server are implemented as a set of Java library classes. The user, location and session objects are high-level objects served by the API. Connection management and event transportation from the MetaWeb client to the server is performed using a special "Connector" object. Incoming events are handled at the client by an *EventManager* while an *ObjectManager* supports the transparent synchronisation between the local object repository and objects in the server. All events are transported using a technique of object serialisation (they are written directly to the network stream) and therefore the body of all events in MetaWeb (an object) must support this. Events also contain a sequence number which provides simple message ordering and concurrency control.

Enhancement of the Web browser (Figure 4) to present and provide access to the new functionality offered by MetaWeb is provided through the integration of Java applets into a standard Web page. This functionality allows the application-specific code to be fetched transparently from a Web server and run when a Web page is loaded into the browser. Although our approach does conflict with the desire not to change the Web, requiring a browser which supports Java, by far the most commonly used Web-browsers, Netscape and Internet Explorer, *do* provide this support across all major operating systems (and future browsers, in order to be accepted, have to follow this de facto standard). Therefore the Java-based approach is preferable over other techniques for enhancing the Web client, such as separate applications which communicate with the browser (using OLE etc.), as these are platform specific and require an additional application to be installed by the user.

5. Examples of MetaWeb in use

In order to demonstrate the viability of MetaWeb for both new and existing applications, this Section presents a new application, CONAVIGATOR, and the extension of an existing Web-based CSCW application, BSCW, which uses MetaWeb to augment its existing asynchronous support.

5.1. A new application: The CONAVIGATOR

CONAVIGATOR is an application built entirely on top of MetaWeb and runs side-by-side with an existing browser (in the current implementation Netscape). It provides presence awareness of other people who are browsing the Web at the same time and place (page), without requiring any changes to these pages - allowing add-hoc cooperation to take place whenever people meet.

The CONAVIGATOR is built as a single Java applet which connects to MetaWeb using the "connector" object. The applet uses a mapping of one MetaWeb location to one page. The applet obtains the current page a user is browsing through the use of Netscape's LiveConnect architecture, and when the applet observes the user navigating to a new page[1] it changes the user's MetaWeb location. At each new location the applet queries MetaWeb about any sessions on that location and presents these to the user in its interface. A MetaWeb event is triggered whenever a user moves to a new page (location) and joins any session at that page. Consequently, other CONAVIGATOR applets browsing the page implicitly receive notification of the new user directly from MetaWeb.

While the sessions at each location are maintained by MetaWeb (description, membership etc.), the applet applies its own application-specific semantics to the sessions. In the current implementation, the CONAVIGATOR uses MetaWeb sessions for coordinating chat, vote and shared whiteboard applications, and for sending application-specific events for these applications (such as the line of text a user has typed into a chat session).

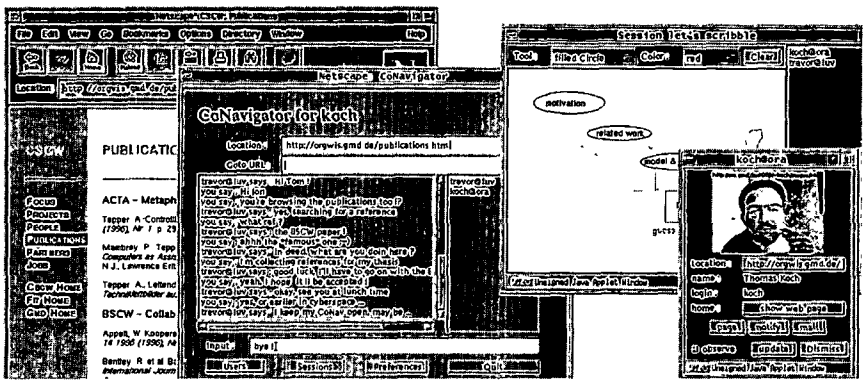

Figure 5 The CoNavigator

The CONAVIGATOR applet is started when a user submits a form from his browser. This form only requires a user name (no registration or authentication is performed) and users may add or remove additional information about themselves at any time later. Figure 5 shows the CONAVIGATOR in use and a number of cooperative sessions (chat and shared whiteboard) which have been started between the users "koch" and "trevor" on a page "orgwis.gmd.de/publications".

"Search" functionality which interrogates the underlying MetaWeb server extends the awareness between users beyond the single page being browsed, providing information (such as the location) of other CONAVIGATOR users. Once users become aware of the presence of others they can get detailed information about each other, similar to an electronic business card. The CONAVIGATOR supports the automatic establishment of communication between users using this information. For example,

[1] Due to security restrictions in Netscape a special "taint" flag must be set by the user prior to using the browser. This allows the applet to get the current location from the browser, but unfortunately causes Netscape to repeatedly issue warnings when this is done This solution works, but is not ideal (see later for alternatives)

if a user enters his COOLTALK address in the CONAVIGATOR (which can be automatically filled in by the application), another user viewing this information simply clicks on the address and (providing he has set up his browser correctly) COOLTALK is launched and connects to that address automatically.

5.2. An existing application: MetaWeb and the BSCW system

The BSCW system (Bentley et al., 1997) is based on the idea of a "shared workspace" which the members of a group establish for organising and coordinating their work. A shared workspace as realised by BSCW is a repository for shared information, accessible to workgroup members using the normal user name/password authentication scheme. In general, a "BSCW server" (a standard Web server extended with the BSCW system through the CGI) manages a number of shared workspaces for different groups and users may be members of several workspaces, perhaps corresponding to the projects the users are currently involved in. These workspaces are accessible from any standard Web browser.

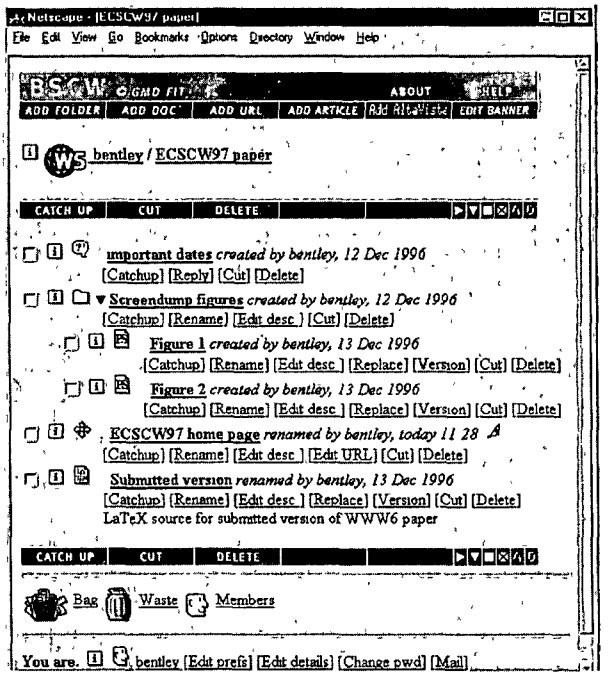

Figure 6 The BSCW system

A workspace can contain different kinds of information, represented as information objects arranged in a folder hierarchy. In Figure 6 the workspace "ECSCW97 paper" contains an article object which holds a simple text message ("important dates"), a folder containing postscript image files ("Screendump Figures"), a URL ("ECSCW97 home page") and a LaTeX document ("Submitted version"). The last "significant" operation performed on each object is described, and a list of clickable "event icons" give information on other recent changes. The operations which can be

performed are given below each object and a description is also presented if one has been supplied (as with "Submitted paper"). Clicking one of the HTML anchors below an object requests operations on the object from the BSCW server (such as "rename"). After each operation the server returns a new HTML page showing the current state of the workspace.

BSCW provides little real-time awareness of the actions of other people co-using BSCW. In practice, a user would not realise that other members of the workspace are interacting with the system at the same time. The only indication a user receives that another user may also be working in a workspace is the event information which is presented about each object. If a user is working with a document, then any changes made to the document will be shown by an event icon in the workspace listing the next time the page is fetched from the server.

The goal of integrating MetaWeb is to extend the BSCW system with continuous feedback of the actions ("activity" awareness) and availability ("presence" awareness) of others. We expect that such awareness will promote a richer form of cooperation to occur, rather than the passive form supported now. There are several issues which need to be considered when extending BSCW with MetaWeb, which are also generally applicable to the extension of any Web-based application:

Relating BSCW concepts to MetaWeb. Workspaces and Sessions

Sessions in MetaWeb define a logical scope for event propagation through a scheme of simple implicit interest registration: an event that is sent to a specific session will automatically be sent to all session members by the MetaWeb server. The location concept allows sessions to be treated together. For BSCW, we want to provide awareness at the granularity of a workspace (which is the highest level at which BSCW provides awareness) for users collaborating within that workspace. Because MetaWeb locations do not have to have a physical or logical equivalent, we can choose to provide one MetaWeb location per workspace. Each of these workspace-locations contains one *BSCW awareness* session plus any additional application-defined sessions supporting synchronous collaboration (e.g. a chat session). The workspace-location and the BSCW awareness session are created and maintained by the BSCW event agent itself (see below), while any other sessions which may occur within that location can be created and managed by the MetaWeb clients themselves.

Integrating the MetaWeb Architecture

Figure 7 shows the BSCW architecture extended with MetaWeb. The underlying BSCW architecture remains unaltered, with the communication between the user's Web browser and the BSCW kernel still based around HTTP/HTML. The extended functionality is achieved by enhancing the Web browser with a MetaWeb client that connects to the MetaWeb event server at the workspace's corresponding MetaWeb location. Information about BSCW events is provided by the BSCW system to MetaWeb through a special *event agent* Upon receiving a BSCW event from the BSCW kernel, the agent puts it into the body of a new MetaWeb event and delivers it to the MetaWeb event server, which is then addressed to the "BSCW awareness" session for the location representing the workspace in which the event occurred. The

event is then distributed as normal and any MetaWeb clients (BSCW users) at the corresponding location (and session) will be notified of the new BSCW event.

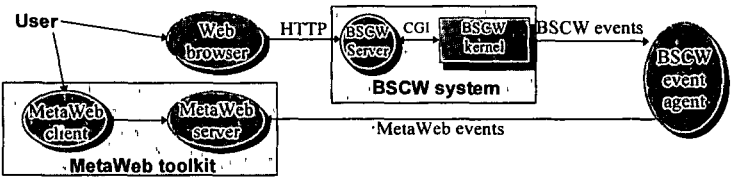

Figure 7 BSCW-MetaWeb integration

Duplicating access restrictions to application information: BSCW events

The BSCW system enforces access control over its objects and events which occur on those objects. In broad terms, the effect of this control is that users can only see events which have happened to objects in workspaces of which they are members. It is important that any provision of synchronous event notification through MetaWeb does not compromise this access control, which would allow viewing of events not otherwise possible. There are two implications of this requirement. Firstly, MetaWeb must restrict access to these events in the same way as BSCW. Secondly, and unlike the CONAVIGATOR, users of the BSCW-MetaWeb clients must be authenticated, i.e. they must be who they say they are.

There are two basic areas where access to events may be restricted: the events themselves (who can receive them) and in the MetaWeb objects (who can join a session). Although both methods are sufficient to mimic the access control provided by BSCW, we have chosen to use the second, session object, method for simplicity. If in the future the event access model in BSCW becomes more complex then the first method can easily be adopted instead.

The problem of authentication is solved using the "hidden" scheme of name-value pair matching (presented in Section 4.4). In BSCW, members authenticate using user names and passwords. Therefore, MetaWeb uses a combination of the user name and password to provide the hidden value to match.

Extending the interface: The BSCW-MetaWeb client

To make use of the synchronous awareness information provided by BSCW and MetaWeb, the workspace user must start a special BSCW-MetaWeb application that connects to the MetaWeb server. The application goes to the appropriate location which represents the workspace the user wants to be aware of, joins the BSCW awareness session at that location, and then receives and displays BSCW events as they occur. This application is built as a Java applet, Figure 8, launched when a user clicks on a special "Awareness" button which has been added to the standard BSCW interface (Figure 6). User details, maintained in a MetaWeb user object, are obtained from the BSCW system itself and encoded into the applet when it is started.

Figure 8 shows the BSCW-MetaWeb application interface. The "User" tab provides "presence" awareness, showing who else is "in" the workspace and using the MetaWeb client. If the user chooses not be "present" then they will not be seen by other MetaWeb users, but neither can the user see them. Clicking on a user's name

brings up information concerning that user in the MetaWeb system. The "Events" tab gives "activity" awareness, presenting the BSCW events as they occur in the workspace. Whenever any user performs an action in a workspace, such as renaming a document, the event will be displayed in the window. The applet is tightly integrated with the browser and clicking on the object names or locations causes the browser to fetch the object from the BSCW system or the location to be visited. The "Session" tab shows the current cooperative sessions which a user can create or join (which have corresponding MetaWeb session objects), such as chat, whiteboard and voting sessions (see Section 5.1). These sessions and their semantics are managed by the BSCW-MetaWeb application and server rather than BSCW itself.

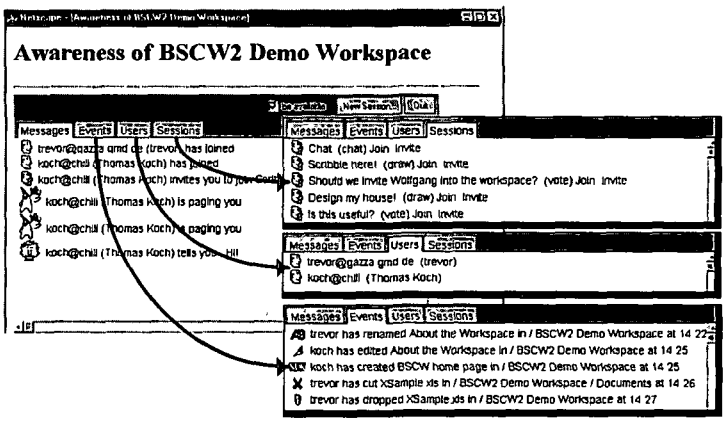

Figure 8. The BSCW-MetaWeb application

A second tool is available which presents an "at a glance" overview of the presence and activities of users in all of the BSCW workspaces the member can access. Information about the changes in the workspaces can be unfolded to a greater level of detail and users can bring up a full MetaWeb awareness client for a workspace (above) simply by clicking on the workspace name.

6. Observations and discussion

The MetaWeb toolkit has been used to develop a variety of small applications on the Web, and two larger ones have been described in this paper. The different uses of Web pages to support cooperation in these applications demonstrates the need and importance of providing flexible cooperative session coupling to the Web, something novel offered only by MetaWeb. In CONAVIGATOR, cooperation takes place in single pages while in the BSCW integration the users' cooperative activities span many pages (which make up a workspace). However, our experiences with MetaWeb have highlighted a number of issues.

Firstly, Section 3.3. presented several goals for a toolkit which were addressed in the design and implementation of MetaWeb. This included the provision of mechanisms to support general access to shared information. In MetaWeb the only

shared objects are those representing the core MetaWeb concepts of user, location and session. While it is possible for applications to define and maintain their own shared objects within the MetaWeb server using the event service, there is no fundamental support for this sharing within the toolkit itself.

Secondly, in Section 4 we explained our motivation to limit the relationship between sessions and locations to deliberately restrict session nesting. It remains to be seen if this simplification in both concept and implementation does not overly restrict other Web-based applications.

Thirdly, the use of a single centralised MetaWeb server may limit the scalability and performance of the current toolkit. For single applications, such as BSCW, this single server approach is not a problem, and Web-based applications tend to service all requests from a single HTTP server anyway (Figure 2). However, a single server approach does not promote inter-operation between different applications using the MetaWeb toolkit (unless they choose to use the same server). For applications which have no centralised coordinating server, such as the CONAVIGATOR, a multi-server approach (with local servers) should boost performance.

Fourthly, while the MetaWeb server guarantees a basic event message ordering, there is no support for such concurrency control in the MetaWeb applications themselves. Consequently, in the example of a chat session, ordering of messages is guaranteed by the client first sending a user's typed chat line to the MetaWeb server, and waiting for the event to be delivered back to itself before displaying the line in the main dialog window. Where more fine-grained or immediate feedback is required, such as with a synchronous shared authoring tool, the existing MetaWeb concurrency control scheme may be too limited.

Finally, this paper has focused on the technical issues and problems of supporting cooperation on the Web. Once that support is present, a number of interesting questions are raised. For example, users of the CONAVIGATOR cannot be sure that users they "meet" are who they say they are. People often ask strangers in the real-world for directions to a place without asking for identification first, and trust those directions they receive. Is it a problem therefore that users of CONAVIGATOR also lack that authentication? Future work with MetaWeb will explore some of the social implications of providing synchronous cooperation on the Web, as well as the more technical issues above.

7. Conclusions

The Web is an important platform for CSCW system development and deployment. Despite the many advantages which make the Web such an attractive platform, developers face a significant shortcoming in the Web's architecture for supporting synchronous cooperation. Changing the Web directly (either its clients or protocols) effects the very aspects which make the Web so useful. This paper has argued that the solution lies in an additional infrastructure which runs alongside the Web.

The MetaWeb toolkit attempts to provide this infrastructure by providing support for the common features found in more general toolkits, such as session management,

access control, mechanisms for accessing shared information, and support for message exchange. Further, the MetaWeb toolkit also addresses the additional requirements which the Web brings: platform and browser independence, variable coupling of application models of cooperation to the Web, generic support for a range of applications, and the support for mutual awareness of others. By supporting both sets of requirements MetaWeb not only aids developers in building and representing a cooperative system on the Web, but also promotes the aspects of the Web which make it such a desirable platform for implementation.

Acknowledgements

The BSCW system reported in this paper is funded as part of the CoopWWW Project by the European Union under the Telematics Applications Programme (TE 2003). We thank Richard Bentley, Wolfgang Appelt, Klaas Sikkel, and David Kerr for their comments and contributions to the paper.

References

Bentley, R., Appelt, W., Busbach. U , Hinrichs, E., Kerr, D , Sikkel, S., Trevor, J. and Woetzel, G. (1997): "Basic Support for Cooperative Work on the World Wide Web" To appear in *International Journal of Human-Computer Studies Special issue on Innovative Applications of the World Wide Web*. Academic Press Available spring 1997.

Berners-Lee T., Caillıau R., Luotonen A , Nielsen H.F. and Secret A. (1994). "The World-Wide Web". *Communications of the ACM,* vol 37 no.8, August, 1994, pp. 76-82

Crowley, T., Milazzo, P., Baker, E., Forsdick, H. and Tomlinson, R. (1990): "MMConf: An Infrastructure for Building Shared Multimedia Applications", *CSCW'90*, October 7-10, Los Angeles, ACM Press, 1990, pp 329-342

Dix, A. (1997). "Challenges for Cooperative Work on the Web: An analytic approach", *Computer Supported Cooperative Work The Journal of Collaborative Computing Special Issue on CSCW and the Web*, Kluwer, 6(2-3), in press.

Ellis C A., Gibbs S.J and Rein, G L. (1991): "Groupware : Some issues and experiences". *Communications of the ACM*, vol.34, no.1, 1991, pp. 38-58

Freund, R. (1996) "A Model to Access Groupware Systems via the World-Wide Web", Masters thesis, Bonn University, Informatik III., 1996. http //orgwis gmd.de/projects/POLITeam/www/

Greenberg, S and Roseman, M (1996): "GroupWeb· A Web Browser as Real-Time Groupware", *CHI'96*, April 13-18, Vancouver, Canada, ACM Press, 1996, pp.271-272.

Grudin, J. (1988) "Why CSCW Applications fail: Problems in the Design and Evaluation of organizational Interfaces",*CSCW'88*, Sept.26-29, Portland, Oregon. ACM Press,1988, pp.85-93.

Jacobs, S., Gebhardt, M., Kethers, S. and Rzasa, W. (1996): "Filling HTML forms simultaneously: CoWeb - architecture and functionality", *5th International World Wide Web Conference WWW'96*, May 6-10, Paris, 1996, pp 1385-1395.

Patterson, J F , Hill, R.D and Rohall, S L. (1990). "Rendezvous: An Architecture for Synchronous Multi-User Applications", *CSCW'90*, October 7-10,Los Angeles, ACM Press, 1990, pp.317-328.

Roseman, M. and Greenberg, S. (1992): "GroupKit A Groupware Toolkit for Building Real-Time Conferencing Applications",*CSCW'92*, Oct 31-Nov 4, Toronto, ACM Press, 1992, pp. 43-50.

Trevor, J , Bentley, R and Wildgruber, G. (1996) "Exorcising daemons. a modular and lightweight approach to deploying applications on the Web", *5th International World Wide Web Conference WWW'96*, May 6-10, Paris, 1996, pp. 1053-1062.

Ubique (1995): "Virtual Places", http.//www.vplaces com/vpnet/index html

Walther, M (1996). "Supporting Development of Synchronous Collaboration Tools on the Web with GroCo", *5th ERCIM/W4G Workshop*, GMD, Sankt Augustin, Germany. Arbeitspapiere der GMD Nr 984, February, 1996, pp. 81-89.

Woo, T K and Rees, M J. (1994). "A Synchronous Collaboration Tool for World-Wide Web", *2nd International World Wide Web Conference WWW'94*, Chicago, 1994