To appear in ACM Conference on Computer Supported Cooperative Work (CSCW) 2000

WebSplitter: A Unified XML Framework for Multi-Device Collaborative Web Browsing

Richard Han, Veronique Perret, Mahmoud Naghshineh
IBM Thomas J. Watson Research Center
30 Saw Mill River Road
Hawthorne, NY 10532 USA
(914) 784-7608
{rhan,perret,mahmoud}@us.ibm.com

ABSTRACT

WebSplitter symbolizes the union of pervasive multi-device computing and collaborative multi-user computing. WebSplitter provides a unified XML framework that enables multi-device and multi-user Web browsing. WebSplitter splits a requested Web page and delivers the appropriate partial view of each page to each user, or more accurately to each user's set of devices. Multiple users can participate in the same browsing session, as in traditional conferencing groupware. Depending on the access privileges of the user to the different components of content on each page, WebSplitter generates a personalized partial view. WebSplitter further splits the partial view among the devices available to each user, e.g. laptop, wireless PDA, projection display, stereo speakers, orchestrating a composite presentation across the devices. A wireless PDA can browse while remotely controlling the multimedia capabilities of nearby devices. The architecture consists of an XML metadata policy file defining access privileges to XML tags on a Web page, a middleware proxy that splits XML Web content to create partial views, and a client-side component, e.g. applet, enabling user login and reception of pushed browsing data. Service discovery finds and registers proxies, browsing sessions, and device capabilities. We demonstrate the feasibility of splitting the different tags in an XML Web page to different end users' browsers, and of pushing updates from the browsing session to heterogeneous devices, including a laptop and a PDA.

Keywords

Collaboration, co-browsing, multi-device, wireless, PDA, proxy, groupware, XML, service discovery, middleware, remote control, pervasive, partial view

1. INTRODUCTION

Collaborative Web browsing with multiple users combines the traditional field of collaborative or conferencing groupware, also known as computer support for cooperative work (CSCW), with the Web. Collaborative Web browsing, also called co-browsing, has been demonstrated commercially by WebEx.com [WebEx], www.uGoCo.com, www.icq.com, www.crowdburst.com, www.n2g.com, www.netdive.com, and Netscape Conference [Netscape], as well as academic projects such as UNC's CobWeb [Stotts97]. RealNetworks' Real Presenter allows multiple recipients to view a PowerPoint presentation on the Web [Reala]. Microsoft's NetMeeting [Microsoft] allows participants to collaborate and share Windows documents in real time over the Internet.

WebSplitter adds two themes to collaborative Web browsing: creating personalized partial views of the same Web page based on each user's access privileges; and creating a composite view of a Web page spread across the set of multimedia devices accessible to each user. The creation of partial views and multi-device composite views is based on the observation that XML tags can be selectively filtered on an individual tag granularity to create sub-documents or partial views. The same XML-based mechanism can be used to create a partial view based on user identity and/or device characteristics. A partial view of an XML page is first constructed based on a user's access privileges to the content in the page. If a user has access to multiple devices, then the partial view is further subdivided into separate XML components that are then delivered to each of the user's locally available devices to form a composite presentation. WebSplitter provides a unified XML framework enabling both multi-device and multi-user Web browsing.

A key design goal of WebSplitter is to enable the presentation of different partial views of the same Web page to different users. We wish to remove the restriction that every member of the audience receives the same

identical Web page as the presenter. To the authors' best knowledge, the previous examples of collaborative Web browsing reproduce the same Web page on each user's browser. Presenting different partial views of the same Web page is helpful in multiple ways. First, if a lecturer has private reminder notes annotating each slide in a Web presentation, then the lecturer may wish to receive the private notes for each slide and yet also wish to prevent others in the general audience from receiving these notes. Second, each Web page in the slide show may contain

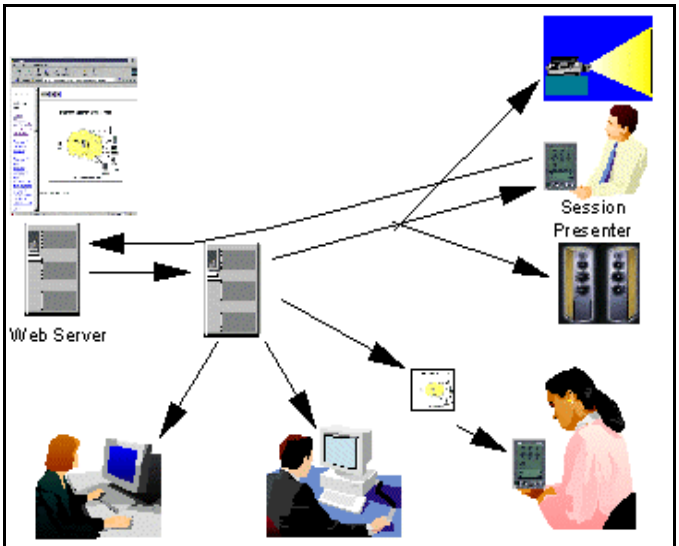

Figure 1. The output of a multi-device collaborative Web browsing session is split to multiple output devices owned by a single user (upper right), as well as to multiple different users, all following the Web browsing of a session presenter (upper right).

forward/backward navigation buttons that the lecturer wishes to prevent the audience from receiving; otherwise, the members of the audience could change the course of the lecture against the presenter's wishes. Third, a lecturer controlling a Web presentation via a wireless PDA would not wish to impose the PDA's representation of the Web page on the other members of the audience. Fourth, partial views can be used to enable different levels of customized instruction for different pupils following the same instructional Web browsing session. These different levels of access privilege can also be used to implement parental control levels.

A second key design goal of WebSplitter is to enrich the browsing experience of a resource-limited device, such as a wireless PDA, by allowing that device to take advantage of the multimedia capabilities of other devices in the vicinity. Ubiquitous computing devices [Want95], e.g. a wireless PDA, wearable computer, a WAP-enabled cell phone, may have a small screen and may lack other features, e.g. audio output. The customary assumption in these examples is that the output results for either information access or peer-to-peer communication are returned only to the single mobile device that requested the information. To address the limitations of mobile devices, prior work has proposed that information content be adapted to the constraints of the mobile device, e.g. transcoding of Web pages for display on PDA's [Fox98, Han98].

In this paper, the limitations of the mobile device are compensated for in a different manner. Rather than confine the output results of an information query to the lone requesting mobile device, WebSplitter enables the output of Web browsing to be spread or split across multiple devices with stronger multimedia capabilities. For example, a wireless PDA that lacks audio output is enabled by the WebSplitter application to exploit a stereo speaker in the vicinity for playback of the audio component of each downloaded Web page. Similarly, if there is a projection display in the room, then the WebSplitter application enables the PDA to redirect part of the output of its Web browsing session onto the large display. In addition, WebSplitter may send a transcoded version of each Web page to the PDA for remote control purposes. Since each Web page is in effect being split for presentation to multiple output destinations, we have named our design WebSplitter.

Figure 1 illustrates integrated multi-user multi-device Web browsing. The session presenter in the upper right requests a Web page whose contents are then split to multiple participants in the joint browsing session. Collaborative browsing is useful for a variety of Web-based presentations, e.g. a teacher presenting a Web-based lecture to multiple students, or a CFO presenting a quarterly financial report to a group of stock analysts on a conference call. Personalized partial views are constructed for each individual. For example, the user with the PDA in the lower right receives only the graphical slide portion of the page (in fact, only a transcoded version) rather than the complete view of the Web page shown in the upper left. The ability of WebSplitter to orchestrate a composite presentation across multiple devices is shown in the upper right. Multiple components of the original page are split for simultaneous presentation and/or playback on a nearby projection display and stereo system, in addition to the requesting PDA. In this way, the capabilities of the wireless PDA are extended, and the session presenter can be viewed as using the PDA as a remote control for the entire browsing session.

The author of a Web page guides WebSplitter in the creation of partial views through the mechanism of a metadata "policy file". Accompanying each Web page is a

separate policy file that contains the rules for mapping hypertext tags, possibly aggregated, to "privilege groups". A policy file may define multiple privilege groups for a given page. For each privilege group, the policy file lists the hypertext components in the Web page that each privilege group may receive. Each individual who subscribes to a collaborative browsing session provides some identification to the WebSplitter application, by password or otherwise, that classifies the individual as belonging to a particular privilege group. By assigning different subsets of hypertext components to different privilege groups, the author can create partial views of the same Web page for different members observing the same collaborative Web browsing session.

A service discovery middleware component provides the abstraction necessary both to enable WebSplitter to discover devices, and conversely to enable users to discover the WebSplitter application. WebSplitter must be able to discover users and their associated output devices so that Web content can be split to the proper destinations. Protocols such as SLP (Service Location Protocol) [Veizades97], Jini [Waldo99], Microsoft's Universal Plug and Play [UPNP], and others [Salutation] [Bluetooth], have been introduced to solve the problem of device discovery. Conversely, each user must be able to discover the WebSplitter application and ongoing browsing sessions. Session discovery solutions are addressed by standard conferencing groupware. These issues are further discussed in Section 3.

In related work, the VIEW Media conferencing system creates partial views of the same hypertext document based on access rights of individual users [Yokota99]. VIEW Media shares our view that a hypermedia document consists of multiple components. VIEW Media appears to achieve partial views by hiding data via overwriting, rather than splitting and filtering Web content as we have proposed. The authors' approach does not appear to be based on XML, nor to address the notion of splitting Web content to multiple devices. The CAT (collaborative active textbooks) system allows multiple users to see different partial views of the same Web animation show [Brown96]. The system is specific to Web pages supporting the animation application.

Prior work on remote control of applications via a PDA has focused on controlling a video conferencing application [Hodes99a] as well as providing a framework for controlling lights and stereo components [Hodes99b]. Our work shares the theme of using XML to enable remote control but differs in our focus on Web browsing, simultaneous presentation across multiple devices, and creation of per-user partial views. In CMU's Pebbles project, a PDA is used to control a single PC's screen and various PC applications, including PowerPoint as well as a Web browser [Myers98]. Our work differs in our emphasis on controlling multiple output devices. Stanford's Interactive Workspaces project [IWork] investigates user interaction with multiple devices and sensors in a room, but to our knowledge has yet to publish their work. Multi-device user interfaces have been proposed in which a user can "pick" an object from a PDA and "drop" it on a PC screen or digital whiteboard [Rekimoto98]. Our work differs by focusing on the Web, and by enabling a PDA to simultaneously control multiple devices. Other work has focused on partitioning of a user interface between a PDA and a single other device [Robertson96].

In the remainder of the paper, we describe the architecture of WebSplitter as well as our implementation experience. The collaborative multi-device Web browsing architecture consists of a server-side metadata policy file (Section 2), a middleware proxy (Section 3), and a client-side applet (Section 4). Section 5 summarizes the progress of our implementation, and our experience with the feasibility of Web splitting.

2. SERVER-SIDE AUTHORING OF XML WEB CONTENT AND THE XML POLICY FILE

.

2.1 The XML Web page

The full capabilities of the multi-device Web browsing proxy will only be exercised when the Web page is written in XML. XML is highly appealing due to its flexibility. Each author defines their own tags. This allows WebSplitter to split the XML document at the granularity of tags, grouping tags into independent components and sub-documents that can be sent to selected devices. While HTML can be also be split by the multi-device Web browsing proxy, tag-granularity splitting of the content is not a realistic option. HTML tags are standard and rigid: the user cannot give meaning to those tags. For example, if the user wants to include two images in a document, HTML only provides the IMG tag to do so, even if the two images have totally different purposes. In a slide presentation, the picture representing the navigation bar and the picture representing the slide have two different roles in the structure of the document; the navigation bar is for control while the slide is the actual data to be displayed.

Our intent is for each uniquely named tag on a page to have its own demultiplexing policy describing the output devices to which the tag may be delivered. We rely on XML's ability to uniquely define the name of each tag, and on XSL to define the usage of each tag. An XSL document is needed by an XML-enabled browser to transform the customized XML into a language that a browser can render, e.g. formatted XML, HTML, WML.

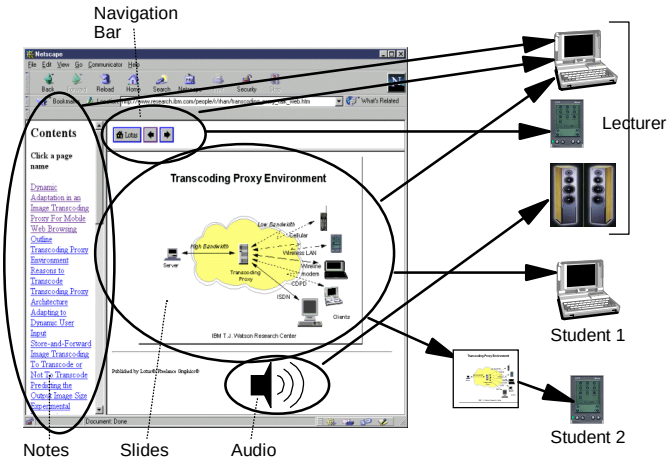

Figure 2. XML tags/components in a Web document (e.g. notes, navigation buttons, graphical slide, and audio) are separated and delivered to different users, according to user preferences and access privileges, and then are further subdivided for delivery to different devices owned by a single user (e.g. session presenter).

```
<?xml version="1.0"?>
<?xml:stylesheet type="text/XSL" href="present.XSL"?>
<presentation>
    <head>
        <title>The Transcoding Proxy</title>
            <nav_bar xml:link="simple"
                href="nav_bar.gif"/ >
    </head>
    <slide>
        <picture xml:link="simple" href="transco.gif"/>
    </slide>
    <notes>
        text...
    </notes>
</presentation>
```

Figure 3: A sample XML Web page from the Web slide show presentation of Figure 2. The three main tagged components on the page are the navigation buttons (<nav_bar>), the graphical slide (<slide>), and the private notes (<notes>).

Consider the following example of the XML Web page for a group slide presentation as shown in Figure 2. A lecturer has authored a Web presentation that is stored on a Web server and desires to show this presentation to other users, perhaps in a telephone conference, conferencing room, or lecture hall. The lecturer possesses a wireless PDA and a laptop. The lecturer desires to navigate the presentation via a wireless PDA, in order to have freedom to roam. Simultaneously, the lecturer wishes to have the graphical slides appear on the lecturer's laptop so that they may be projected on to the lecture hall's large screen. Also, lecturer wishes to hide reminder notes and navigation buttons from the audience. In this case, the navigation bar with forward/back buttons and private notes should be sent to the lecturer's PDA. Only the graphical slide should be sent to student members (possibly transcoded if students own PDA's). There are four components to the Web page shown in Figure 2: navigation buttons; graphical slides, private notes; and audio. The XML file used to describe the non-audio components of such a page could look like Figure 3. The document has a header portion with a title (tag <*title*>) and a navigation bar (tag <*nav_bar*>) and two body portions containing the slide (tag <*picture*>) and private notes (tag <*notes*>). We have not shown the accompanying XSL file.

2.2 The XML policy file:

In addition to the XML Web page and XSL style sheet, the author may optionally write a metadata *policy file* that annotates the original XML page and defines mapping rules governing which XML tags should be distributed to which privilege groups and which devices. The policy file provides the information that enables the WebSplitter application to determine what portions of a Web presentation may be seen by which users. The policy file defines privilege groups, and then defines what subset of XML tags each privilege group is permitted to receive. The author can use the policy file to limit certain groups to receiving only a partial view of the Web presentation, while granting full access privilege to other viewers.

Privilege groups provide a mechanism for differentiating the access privileges of participants. *Partial views* of a Web browsing session belong to different privilege groups. The author of a Web page may wish to give some user groups higher or lower privileges to access certain XML tags than other groups. Figure 4's sample policy file realizes the mappings desired in Figure 2. Two groups are defined: lecturers and students. A lecturer who is collaboratively presenting an e-class limits students to seeing only a subset of the tags on each presented Web page, i.e. the lecturer prevents the students from seeing private reminder notes (<notes> tag is not available to students), and prevents the students from accessing the navigation controls to the lecture (<nav_bar> tag is not available to students).

```
<cmdb:group name = "lecturer" password = "YYYYY"
input_permitted = "yes">
    <cmdb:device name = "IE||Netscape">
        <cmdb:taglist> presentation, head, title, nav_bar,
            slide, picture</cmdb:taglist>
        <cmdb:undefinedtag_list>images, text, audio,
video<cmdb:undefinedtag_list>
    </cmdb:device>
    <cmdb:device name = "PocketIE">
        <cmdb:taglist>presentation, head, title, nav_bar,
            notes</cmdb:taglist>
        <cmdb:undefinedtag_list>text</cmdb:undefineda
g_list>
    </cmdb:device>
</cmdb:group>

<cmdb:group name = "students" password = "ZZZZZ"
input_permitted = "no">
    <cmdb:device name = "IE||Netscape">
        <cmdb:taglist> presentation, slide, picture
            </cmdb:taglist>
        <cmdb:undefinedtag_list>images, text, audio,
            video <cmdb:undefinedtag_list>
    </cmdb:device>
    <cmdb:device name = "PocketIE">
        <cmdb:taglist> presentation, slide,
            picture(transcoded )</cmdb:taglist>
        <cmdb:undefinedtag_list>text
            </cmdb:undefinedtag_list>
    </cmdb:device>
</cmdb:group>

<cmdb:media_types>
    <cmdb:images>
        <cmdb:taglist>nav_bar, slide, picture</taglist>
    </cmdb:images>
    <cmdbtext>
        <cmdb:taglist>title,notes</cmdb:taglist>
    </cmdb:text>
</cmdb:media_types>
```

Figure 4: A sample XML policy file defining multiple privilege groups and the tags permitted to be received by each privilege group, using the tags defined in Figure 3.

The policy file is written in XML syntax. The author benefits by not having to learn a new syntax, and the WebSplitter proxy benefits by being able to reuse its XML parser for Web pages to interpret policy files. The XML syntax within the policy file must be standardized, i.e. we need to define our own XML namespace or schema to standardize the tags and values used in those policy files. In the following, we propose to use the namespace prefix cmdb (collaborative multi-device browsing). All authors should use the standard tag name "group" in order that all WebSplitter proxies understand that a privilege group is being defined. Our sample XML policy file includes the following proposed standard tag names: group, device, taglist, undefinedtag_list, media_types, images, and text (video and audio not shown). In addition, each tag has its own unique attributes, e.g. the group tag has the attributes "name", "password", and "input_permitted" (explained later). Finally, while the value of the attributes "name" and "password" need not be standardized, the values of certain attributes must be standardized, e.g. input_permitted must have standardized values set to either "yes" or "no" -- otherwise different authors will use true/false, on/off, 1/0. The naming of devices also needs to be common over devices, so that the proxy can universally associate capabilities with a device name.

We introduce an attribute *capability* to the device tag to let the author specify the capabilities of the device. For example, the author writes *<cmdb:device capability="sound_card">*, meaning that for any device that has a sound card, the following set of tags can be sent. It is even more crucial here to standardize the values of the *capability* attributes so that the proxy can match them with what it retrieves from service discovery. The proxy will learn the capabilities of the devices from the various attributes registered in the service discovery database for each device. If there's no standard vocabulary to describe the capabilities, we cannot interpret this information. The W3C draft "Composite Capability/Preference Profiles (CC/PP)" has followed the approach of globally standardizing the characteristics of devices [W3C].

In order to determine whether or not an XML Web page has an accompanying policy file, the proxy must parse the XML page that has been retrieved. If the proxy finds a reference to the policy file, then the WebSplitter proxy must fetch the policy file. The mechanism for embedding the policy file reference in the XML follows the method for referencing a stylesheet document from XML An XML document indicates which stylesheet to use with the following type of processing information, *<?xml:stylesheet type="text/XSL" href="present.XSL"?>*. In the same way, we propose to use a processing instruction like *<?xml-cmdb href = "location of policy file"?>*. The multi-device Web browsing proxy recognizes this instruction and retrieves the file located at the address specified in the *href* attribute.

The input_permitted tag controls which users have the right to change the Web page shown to the rest of the browsing members. The mechanism was added to protect the session presenter from audience members that otherwise could maliciously change the course of the presentation. This is a crude form of floor control, and we have not pursued this aspect of the design further.

In several cases, undefined tags will be encountered by WebSplitter. We have incorporated the tag *undefined_tag* into the policy file's syntax to provide an indication about how unknown tags are to be mapped to devices. The tag *undefined_tag* specifies the list of media types that are permitted to be sent to each combination of privilege group and device. This form of media-based splitting is suboptimal, but the proxy has little option other than to fall back to this coarse-grained form of page splitting. Undefined tags will be encountered when an HTML document is accessed, or an XML document containing tags not listed in the original policy file is encountered.

3. MULTI-DEVICE WEB BROWSING PROXY

WebSplitter's document-splitting function is placed in a proxy. The proxy utilizes the policy and XML files written by the author to establish a multi-device browsing session and control the ad-hoc collaboration between devices. The proxy can flexibly reside anywhere in the network, i.e. near the server, intermediate between client and server, in the same room as the user, or even on the user's body. The proxy also has the capability to limit the processing conducted on thin clients, which only need to run a small applet-like program to participate in a joint browsing session. The proxy can also perform prefetching of files (images, audio files…) without waiting for an end-user to request them, thereby improving responsiveness.

3.1 Establishing and joining browsing sessions

Figure 5 summarizes the steps involved in establishing a multi-device collaborative browsing session in a WebSplitter proxy. The proxy registers itself (1) to the service discovery database so that a client can find the proxy (2). Next, the session creator clicks on the desired proxy's hyperlink (3) and the proxy returns a session login menu requesting the session name and URL of the first XML Web page (4). Then, the proxy pulls the requested XML page (5) and its associated policy file (6) into the proxy. The proxy parses the XML page and policy file to create privilege groups (7), then sends to the session creator a login menu requesting username and password (8). After matching the session creator to a privilege group, the proxy announces the new session to service discovery and requests any devices registered by the session creator (9). Finally, the portion of the requested XML page that the session creator is permitted to receive is returned to the session creator (10).

Figure 6 summarizes the steps involved in joining an ongoing browsing session. Each new device that wishes to participate in the session first registers its capabilities and its owner to the service discovery database (SDDB), and

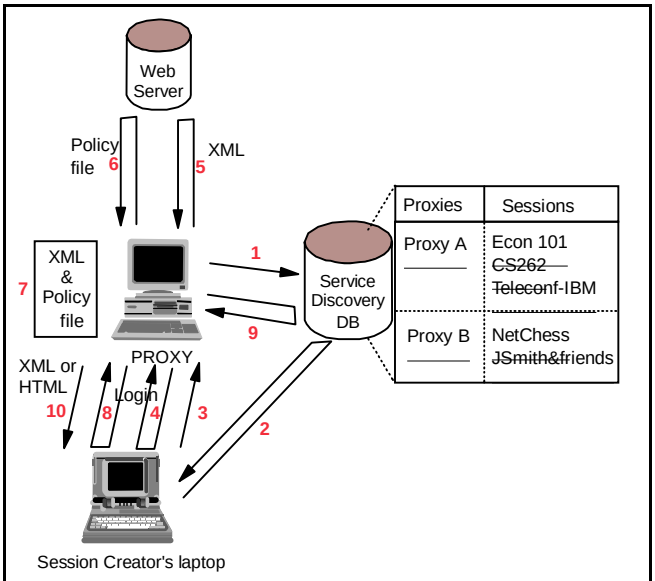

Figure 5: Procedural steps for establishing a session of collaborative multi-device Web browsing.

then queries the SDDB to discover existing proxy sessions (11, 11', and 11'' (advertise only)). Next, each browser-enabled device connects to the proxy (12, 12'), and logs in for the session (13, 13'). User login triggers the proxy to contact the SDDB to discover all devices registered to the newly logged-in user (14, 14'). Finally, the proxy demultiplexes the appropriate Web objects to the

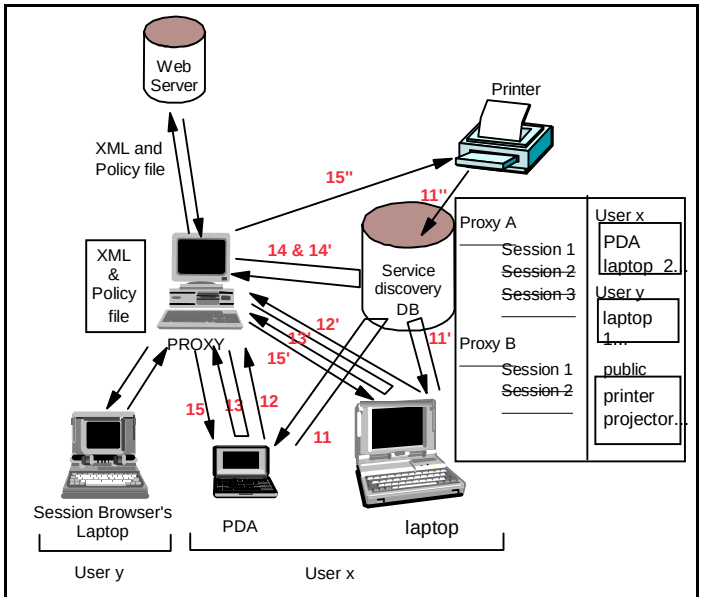

Figure 6: Procedural steps for joining an existing collaborative multi-device Web browsing session.

appropriate end clients/users (15, 15', 15'').

3.2 Building tag mapping rules from the policy file

At session establishment, the proxy fetches the Web page specified by the session creator. If there is a reference to a policy file in the parsed Web page, then the proxy downloads and parses the XML policy file from the Web server. The policy file specifies what subset of XML tags each privilege group and device is permitted to receive. At this point, the proxy only has general information on what tags each group is allowed to receive. It has not yet built any policy for mapping tags to specific users. In order to build a complete mapping, the proxy still needs each user to login via password and identify themselves as a member of a given privilege group. Each subscribing user indicates their group membership during login either by providing a password or by selecting a non-password-protected group. The policy file defines a unique password for each privilege group. When the user provides a password, it is matched to a unique privilege group. We assume that the password is known to the user in advance via some out-of-band method. Combining those two sources of information, the proxy will dynamically build rules that map permitted tags to users.

As a final refinement of the tag mapping rules, the proxy needs to map XML tags to each of the multiple devices accessible to each user. The proxy queries service discovery to retrieve the list of registered output *devices* owned by a particular user. This list contains the access method required by each device, so that the proxy knows the multimedia format and communication protocol with which to send Web components to each specific device.

3.3 Generating receiver menus, partial views, & panels

A session creator receives two login screens, a session login menu (Step 4) and a user login menu (Step 8). A subscriber to an existing session receives only the single user login menu (Step 13). The session login menu asks for the name of the session to be established and the URL of the first page to be downloaded. This URL indirectly specifies the initial policy file used to define privilege groups for the session. Using the information found in the policy file, the proxy builds a customized user login menu. The user is asked to enter a username and password. These two fields are used by the proxy both to identify the privilege group of the user/subscriber and to find registered devices. The proxy queries the SDDB for a list of output devices registered under that username, so the provided username must match the registered username. If there is a difference in usernames, then the proxy will be unable to associate the devices with their proper owner. At this point, the proxy can build the mapping of XML tags to devices. This internal mapping is also exposed to the end user for customization, as discussed later in Section 4.2.

Some privilege groups can be openly accessible without passwords. For example, we wish to allow unknown audience members to subscribe to public presentations, e.g. press briefing, without requiring a priori knowledge of the password. Public non-password-protected groups are shown with a checkbox next to them in the user login menu. The password defined in the policy file is encrypted, as is the response to the login menu, to ensure security.

Following user login and retrieval of the user's list of devices, the proxy will have sufficient information to construct mappings of XML tags to a user's devices. The tag mapping rules combine to form a sub-document or partial view of the original Web page. The proxy needs to construct the appropriate transformation file, e.g. XSL style sheet, that transforms XML to a renderable description like HTML, WML, or even a non-browser device like a speaker. The transformation file built for each client should take that into account to produce a sub-document in the right format. We leverage XSL's power to select tags in the original XML to lay out the sub-document for ultimate presentation on the user's device.

In addition to the partial view, a control panel is also downloaded to browser-enabled devices. For those supporting multiple windows, e.g. standard browsers, WebSplitter opens two windows at the receiver, one for displaying the partial view of a Web page and a second for displaying a control panel. The control panel contains multiple buttons. First, a "logout" button can be pushed to inform the proxy to remove this user from the joint browsing session and delete all state, e.g. transformation files and tag mapping rules, associated with that user. Second, a "personalization" button can be pushed that pops up a third window with the menu shown in Figure 7. This third window enables the user to customize how Web content is demultiplexed to the user's various devices. We are investigating a solution for PDA's that doesn't support multiple windows.

3.4 Interacting with service discovery

Service discovery is used throughout the design, in Steps 1, 2, 9, 11, and 14. WebSplitter adopts the model that service discovery operates as a single abstraction that provides the means for discovery and registration of proxies, sessions, as well as users' devices. Each WebSplitter proxy registers itself and its ongoing browsing sessions to the abstraction. Each output device registers itself and identifies its owner to the abstraction. Users desiring to establish a new WebSplitter session query the abstraction to find a list of registered proxies. A user interested in subscribing to an

ongoing joint browsing session sends a query to the abstraction to find existing WebSplitter sessions. The proxy queries the abstraction to retrieve the list of devices owned by each user in the session. The service discovery abstraction essentially provides an API to WebSplitter that supports two functions, namely register() and query(). The abstraction isolates our application from the details about how these two functions are implemented. Similarly, the application need not know whether the service discovery registry, or SDDB, is concentrated in a single server or distributed among multiple servers. The WebSplitter application relies on service discovery to provide geographic scope and locality. Wide area support will permit distance learning by remote students, who are able to discover the collaborative browsing sessions, and whose devices are in turn discovered by the proxy after login. For our initial prototype, we have deferred wide-area questions and assumed that only users who are within close proximity to the WebSplitter proxy will be able to take advantage of the Web-splitting functionality.

4. CLIENT-SIDE FUNCTIONS

4.1 Pushing data to the client

Each time a session presenter requests a new page, this update must be pushed to all subscribed browsers and devices, e.g. speakers or TV screen, so that each subscriber is synchronized with the joint presentation. Ideally, we would like to avoid modifications to client-side code and leverage existing browser technology. Two approaches could be used to push data to the end browser: server push or client pull. In server push, the server sends down a chunk of data; the browser displays the data but leaves the connection open; whenever the server wants, it sends more data and the browser displays it, leaving the connection open. In client pull, the server sends down a control message, possibly a directive (in the HTTP response or the document header), that says "reload this data/URL in 5 seconds". After the specified amount of time has elapsed, the client does what it was told -- either reloading the current data or getting new data. The continuous polling overhead of pulling clients makes server-push more efficient.

Java provides a push solution for Netscape, IE and more generally for all Java-enabled browsers: it is to use an applet which will listen for notification from the proxy and tell the browser to show a document with the URL sent by the proxy. This approach is a hidden client pull but made more flexible by the presence of the applet which is notified of when new data is available. The page is not refreshed at regular intervals, but only when new data is available. Rather than writing different client-side code relying on vendor-specific push solutions for Netscape (plug-ins) or Microsoft (Active X), we write one applet capable of implementing data-push on any Java-enabled browser. We have not found a universal push solution for browsers that don't have Java capabilities. This is the case of most PDA browsers: Pocket IE for WinCE devices or browsers like Palmscape or HandWeb for Palm Pilots. WAP-enabled

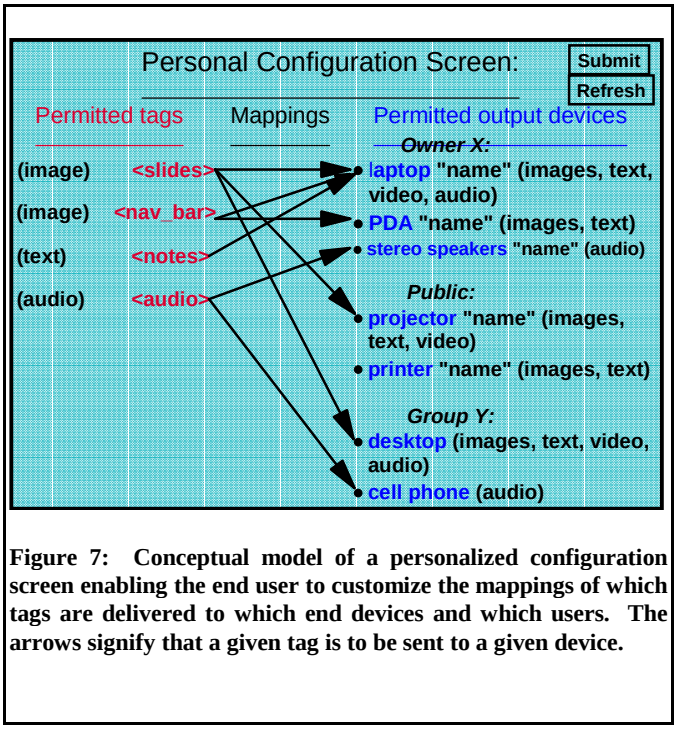

Figure 7: Conceptual model of a personalized configuration screen enabling the end user to customize the mappings of which tags are delivered to which end devices and which users. The arrows signify that a given tag is to be sent to a given device.

WML browsers intend to support data push [WAP], but this technology is still in the development stage. For WinCE devices, we have implemented an HTMLviewer control. An HTMLviewer control relies on a library provided by Microsoft. It implements version 3.2 of HTML and is the basis of Pocket IE. Our control is able to accept data push, receive Web content and display it on the WinCE device.

4.2 Customizing the XML tag mapping rules

The default mapping of permitted tags to permitted output devices generated by the proxy may not satisfy the end user. Therefore, WebSplitter exposes this default mapping to the end user to allow the user to customize where Web components are sent and displayed. For browser-enabled devices, a customization applet is downloaded to enable the user to change the distribution of components among the devices. Conceptually, each tag is permitted to be sent to a list of permitted output devices, as shown in Figure 7. The user can choose to overrule default mappings and redirect some components to another device, or redesign the sub-documents created at the proxy. In practice, rather than redirect graphical arrows, a nested hierarchy is shown in which each output device has a list of tags that are being sent to that device. In the spirit of ad-hoc collaboration,

new devices can be added to and removed from the list of permitted output devices in Figure 7. Pushing the "refresh" button signals the proxy to query for any devices that have entered/left the vicinity. This allows the user to register a device after joining an ongoing session without losing the customization choices already made. The personal configuration screen is invoked within the control panel mentioned in Section 3.4.

5. IMPLEMENTATION STATUS

Our initial prototype focused on browser-enabled clients. Our implementation is primarily concerned with demonstrating the feasibility of splitting an XML Web page to multiple users as well as multiple devices. We have implemented Steps 3-8 and Step 10 shown in Figure 5, as well as Steps 12, 13, and 15 in Figure 6.

For each user, we generated a partial view in two phases. In the first phase, we generated from the original XML_0 page a "sub-document" XML_1 containing a subset of XML tags. To ease the task of eliminating tags, we used an XSL transformation engine called *xalan*, rather than create our own parser. The policy file is used to dynamically build an XSL file, XSL_1. XSL_1 and XML_0 are then both fed as input into the transformation engine to generate a sub-document XML_1 containing only the tags that the user is permitted to and equipped to receive, i.e. a subtree of the initial tree.

In the second phase, our goal was to produce from XML_1 a new partial view document that was in a format understandable by the intended end device. Again, we chose XSL as the transformation file because it is applicable when the partial view should be in a markup language like WML, HTML, XML or even for a pdf output file. In order to respect the style of the author, this second XSL transformation is based on the original XSL document XSL_0 if it exists. At this point, we run into a problem if we apply XSL_0 directly to XML_1, because some of the tags referenced in XSL_0 no longer exist in XML_1. For example, the XSL could be: use the following text as a link whose URL will be defined from an attribute of the tag T. But if tag T has been removed from the XML, no value will be found and there will be an undesirable line of text in the resulting document. To avoid this problem, we could choose to generate a new XSL_2 (derived from XSL_0) that only referenced the subset of tags contained XML_1. However, this introduces its own problems. XSL can contain relative paths to tags (e.g. ./ or ../). This makes it complex to remove all references to the unwanted tags. Our solution was to apply the original XSL_0 to the subset XML_1, and write the XSL_0 in a special way that linearly referenced XML tags; no relative path is used and we define the action to be taken with each XML tag in a separate template element.

Our current prototype demonstrates the feasibility of splitting an XML Web page to multiple users with different devices, according to rules specified in a policy file. We have written a sample XML Web slide show patterned after Figure 2, as well as the associated XSL and XML policy files. The proxy downloads the XML Web page, parses it, and then requests the policy file, parsing the definitions of multiple privilege groups to generate XSL_1. We have verified that multiple users can log into the proxy and are identified as belonging to different privilege groups. The proxy generates different XSL files for different privilege groups and thereby is able to construct different partial views of the initial document for members of different groups. We have also successfully implemented the data-push protocol via an applet to enable joint browsing from a PDA.

Using a manual method of logging into a WebSplitter proxy, we have been able to demonstrate the splitting of a Web document to several users as well as to several browser-enabled devices: a user having a laptop and a desktop can use the personalization applet to customize the way the Web components are dispatched. Two users belonging to different groups receive different views of the same document. The two pictures of Figure 8 are screen captures of what a subscriber received with the multi-device Web browsing. The first picture shows the screen of a WinCE device that has been pushed via the HTML viewer data-push mechanism a version of the presentation that includes both the navigation bar (originally an image, now transcoded into text links) and the lecture's private reminder notes. The WinCE device has been transformed to act as a remote control for the joint browsing session. The lower picture shows the browser window received by a second user. Only the graphical slide is shown. Consequently, the policy file has successfully designated that the Web objects to which this user doesn't have access must be filtered out, and a partial view of the Web presentation has been generated by the proxy.

Our initial implementation of multi-device Web browsing employs static service discovery. The devices available in the environment are manually pre-stored in the service discovery database. The WebSplitter proxy queries the SDDB to learn about available devices. The proxy registers itself with service discovery. The users that want to establish a session or join an on-going session access the service discovery home page, request the list of available devices and select the WebSplitter service, which redirects the user to the WebSplitter home page. In the future, we plan to add dynamicity to our service discovery so that devices can be discovered or removed from the service discovery database as they come and go.

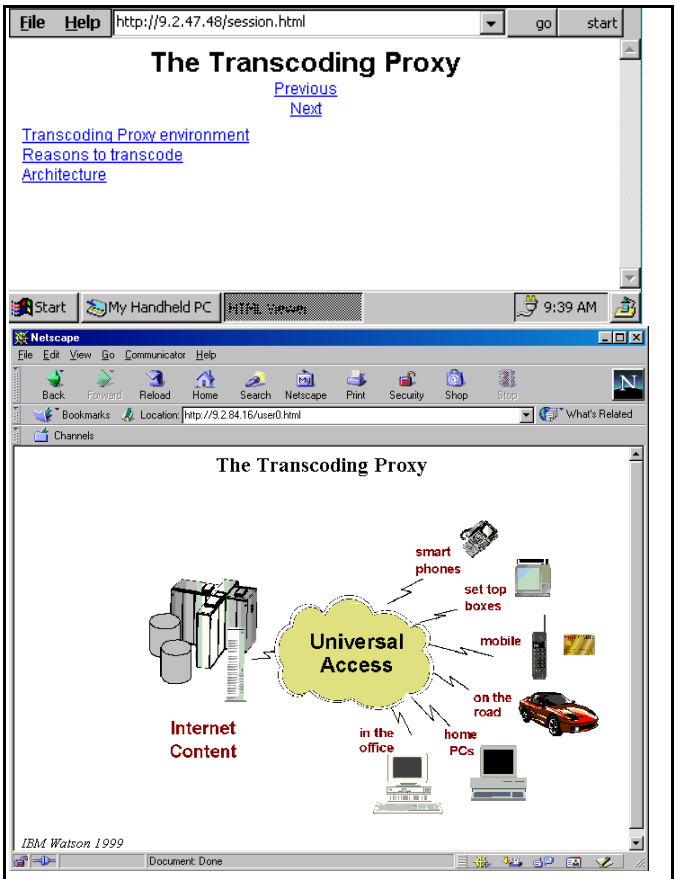

Figure 8: Screen captures of WebSplitter in action. The XML Web page of Figure 2 has been split by our proxy into different partial views on the WinCE screen (top, only the private text notes plus navigations buttons appear, slide is filtered out), and on a standard browser (bottom, only the graphical slide appears, not text or buttons).

6. SUMMARY

WebSplitter enables multi-device multi-user Web browsing via a unified XML framework. The WebSplitter proxy splits XML content to multiple users, constructing personalized partial views based on users' access privileges, as defined in an XML metadata policy file. The partial view is further split into components that are sent to the individual devices accessible to that user. The same mechanism for mapping of permitted XML tags to users is reapplied to map XML tags to output devices. The feasibility of this approach was demonstrated.

7. REFERENCES

[Bluetooth] Bluetooth Serv. Disc., http://www.bluetooth.com */developer/specification/specification.asp*

[Brown96] M. Brown, M. Najork, "Collaborative Active Textbooks: a Web-Based Algorithm Animation System for an Electronic Classroom," *IEEE Symposium on Visual Languages*, 1996, pp. 266-275.

[Fox98] A. Fox, S. Gribble, Y. Chawathe, E. Brewer, "Adapting to Network and Client Variation Using Infrastructural Proxies: Lessons and Perspectives", *IEEE Personal Communications*, vol. 5, no. 4, Aug. 1998, pp. 10-19.

[Han98] R. Han, P. Bhagwat, R. LaMaire, T. Mummert, V. Perret, J. Rubas, "Dynamic Adaptation In an Image Transcoding Proxy For Mobile Web Browsing," *IEEE Personal Communications*, vol. 5, no. 6, Dec. 1998, pp. 8-17.

[Hodes99a] T. Hodes, M. Newman, S. McCanne, R. Katz, J. Landay, "Shared Remote Control of a Video Conferencing Application: Motivation, Design, and Implementation," *SPIE Multimedia Computing and Networking (MMCN), Proc. SPIE*, vol. 3654, 1998 (conf. held Jan 1999), pp. 17-28.

[Hodes99b] T. Hodes, R.Katz, "A Document-based Framework for Internet Application Control," *2nd USENIX Symposium on internet Technologies and Systems (USITS)*, 1999, pp. 59-70.

[HTMLviewercontrol] *http://msdn.microsoft.com/isapi/msdnlib.idc?theURL=/library/wcedoc/wcehpc/jup_prog_29.htm*

[IWork] Interactive Workspaces *http://graphics.stanford.edu/projects/iwork/*

[Microsoft] Microsoft NetMeeting, *http://www.microsoft.com/catalog/display.asp ?site=113&subid=22&pg=1*

[Myers98] B. Myers, H. Stiel, and R. Gargiulo. "Collaboration Using Multiple PDAs Connected to a PC." *Proceedings CSCW'98: ACM Conference on Computer-Supported Cooperative Work*, November 14-18, 1998, Seattle, WA. pp. 285-294. See also *http://www.cs.cmu.edu/~pebbles/*

[Netscape] Netscape Conference, *http://www.aibn.com/help/Software/Netscape/Communicator/Conference/ browsing.HTML*

[Reala] Real Presenter, *http://www.realnetworks.com/products/presenterplus/info.HTML*

[Rekimoto98] J. Rekimoto, "A Multiple Device Approach for Supporting Whiteboard-based Interactions," *Human Factors in Computing Systems (CHI)* 1998, pp.344-351.

[Robertson96] S. Roobertson, C. Wharton, C. Ashworth, M. Franzke, "Dual Device User Interface Design: PDA's and Interactive Television," *SIG CHI* 1996, pp. 79-86.

[Salutation] http://www.salutation.org/

[Stotts97] D. Stotts, J. Prins, L. Nyland, T. Fan, "CobWeb: Tailorable, Analyzable Rules for Collaborative Web Use," *http://www.cs.unc.edu/~stotts/cobWeb/doc/index.HTML*

[UPNP] http://www.upnp.org/

[Veizades97] J. Veizades, E. Guttman, C. Perkins, S. Kaplan, "Service Location Protocol Internet Draft #17," draft-ietf-svrloc-protocol-17.txt, IETF, 1997.

[W3C] *http://www.w3.org/TR/annot/*

[Waldo99] J. Waldo, "The Jini Architecture for Network-centric Computing," *Communications of the ACM*, July 1999, pp. 76-82.

[Want95] Want, R. Schilit, B.N. Adams, N.I. Gold, R. Petersen, K. Goldberg, D. Ellis, J.R. Weiser, M., "An overview of the PARCTAB ubiquitous computing experiment," *IEEE Personal Communications*, vol.2, no.6, Dec. 1995, pp. 28-43.

[WAP] *http://www.wapforum.org/*

[WebEx] *http://www.Webex.com*

[XML] *http://www.w3.org/XML/*

[Yokota99] Y. Yokota, T. Nakamura, H. Tarumi, Y. Kambayashi, "Multiple Dynamic View Support for Cooperative Work," *Proc. 6th IEEE International Conference on Database Systems for Advanced Applications*, 1999, pp. 331-338.