RC 21744 (Log 97704) 03/09/2000
Computer Science

IBM Research Report

WebSplitter: Multi-Device Collaborative Web Browsing

Richard Han
IBM Research Division
Thomas. J. Watson Research Center
PO Box 704
Yorktown Heights, NY 10598

Veronique Perret
IBM Research Division
Thomas. J. Watson Research Center
PO Box 704
Yorktown Heights, NY 10598

Mahmoud Naghshineh
IBM Research Division
Thomas. J. Watson Research Center
PO Box 704
Yorktown Heights, NY 10598

IBM Research Division
Almaden - Austin - Beijing - Delhi - Haifa - T.J. Watson - Tokyo - Zurich

WebSplitter: Multi-Device Collaborative Web Browsing

Richard Han, Veronique Perret,
Mahmoud Naghshineh

IBM T.J. Watson Research Center
{rhan,perret,mahmoud}@us.ibm.com

ABSTRACT

This paper presents an architecture and a corresponding design that enables multiple devices, e.g. laptop, wireless PDA, projection display, stereo speakers, to collaborate in a joint presentation of a Web browsing session. WebSplitter enriches the Web browsing experience of a resource-limited mobile device, such as a wireless PDA, by enabling the mobile device to exploit the multimedia capabilities of other devices in the vicinity, and thereby overcome its own limitations. WebSplitter first splits a requested Web page into different sub-documents and/or media components and then sends these sub-documents/components to a variety of local devices. The collection of devices, ranging from fixed entities such as a large projection display or a stereo system to mobile entities such as a PDA or a Bluetooth-enabled personal area network (PAN), are coordinated via WebSplitter to dynamically create an enhanced composite Web presentation. Among the applications enabled by WebSplitter is that of a wireless PDA acting as a remote control for a browsing session presented over a room's projection screen and stereo system. Since devices may be owned by different human users, WebSplitter extends its design to integrate the concept of joint multi-user presentation, i.e. collaborative groupware or conferencing, with the notion of multi-device presentation of a Web document. The WebSplitter architecture consists of three components: an XML metadata policy file that defines rules governing which tags in an XML Web page can be received by which groups of users in a browsing session; a middleware proxy that splits Web content based on the identity of a user, the capabilities of that user's output devices, and the access privileges granted in the policy file to that user's group; and a client-side component, e.g. applet or other technology, for user login and reception of pushed browsing data. Our architecture incorporates themes from service discovery and user management in order to discover and register proxies, Web browsing sessions, as well as each device's characteristics, e.g. address, capabilities, owner. We describe a preliminary implementation of WebSplitter in which we demonstrate the feasibility of splitting the different tags in an XML Web page to different end users' browsers. We also demonstrate the feasibility of pushing updates from the browsing session, e.g. new Web pages, to heterogeneous devices, including a laptop and a PDA.

Keywords

Multiple devices, collaboration, mobile, PDA, proxy, groupware, remote control, XML, PAN, service discovery, middleware, access control

1. INTRODUCTION

The emerging trend in personal and pervasive computing devices is towards smaller and more lightweight models that the user can carry and operate anytime, anywhere. Such mobile devices, e.g. a PDA, wearable computer, cell phone, are often limited to support only a small set of functions. For example, a PDA may have a small screen and lack audio output, so that only limited display of text and small images is supported. In addition to physical constraints, such mobile devices have limited capabilities when acting in stand-alone disconnected mode. PDA's support highly useful applications such as calendaring and address books. However, the functions that non-networked PDA's can provide are inherently limited to those provided on board the portable device.

Integrating networked connectivity into portable devices enables the user of the mobile device to access a far richer set of applications that may be provided by the network. For example, the PDA with a wireless Internet link may be used to access the user's own desktop applications, email, and the Web, as well as to contact other users for a phone conversation, text chat, etc. The customary assumption in these examples is that the output results for either information access or peer-to-peer communication are returned and confined to the single mobile device that requested the information. Prior work has dealt with the limitations of networked mobile devices by transforming information content to adapt 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, we extend the output paradigm for mobile networked devices to permit these devices to utilize the multimedia output capabilities of other networked devices, both mobile and fixed. Leveraging off of the capabilities of multiple devices beyond the user's limited device holds the promise of enabling the creation of new applications as well as enhancing current applications.

The application that we have chosen to enhance is Web browsing. A key design goal is to enrich the Web browsing experience of a resource-limited mobile device, such as a wireless PDA, by providing functionality that enables the mobile device to exploit the multimedia capabilities of other devices in the vicinity, and thereby overcome its own limitations. 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. In Figure 1, the session presenter in the upper right requests a Web page whose contents are then split into multiple sub-documents that are then delivered to multiple devices (upper right) as well as multiple users. The session presenter is also pictured controlling the browsing session via a PDA in remote control fashion.

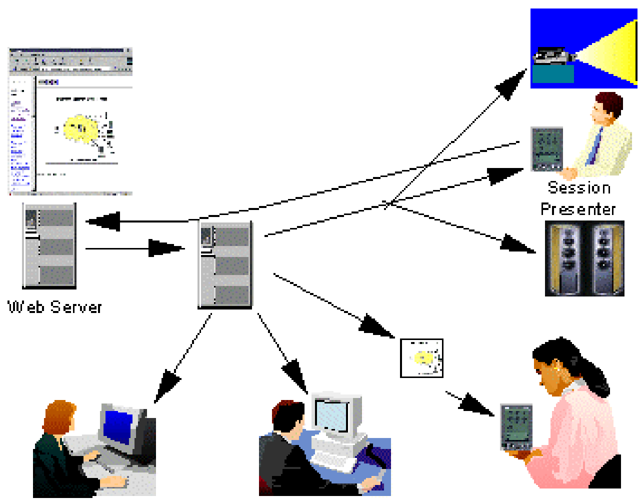

Figure 1. The output of a multi-device collaborative Web browsing session is split to multiple output devices owned by a single user (upper right), as well as to multiple different users, all following the Web browsing of a session presenter (lower left).

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 tight focus on controlling and enhancing the Web browsing application to enable collaborative presentation. 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 from Pebbles in our emphasis on controlling multiple output devices. Stanford's Interactive Workspaces project shares our interest in multi-device browsing and is initiating implementation of an architecture with the goal of enabling multiple input and output devices to collaborate [IWork].

The WebSplitter function can placed either in the backbone network, separate from the user, or in the local environment of the user. The WebSplitter function could operate as a local personal server and reside either on a user's laptop or on a user's wearable computer. In either case, the WebSplitter application acts to control and coordinate joint presentation of a Web document over multiple local devices in the same manner as it would operate remotely in the backbone. The collection of local devices could for example form a Bluetooth-enabled personal area network (PAN) centered on the user's wearable computer with e-goggles as the display and a cell phone for audio playback. Since WebSplitter supports dynamic membership in the joint presentation, then the variety of local devices may also fluctuate as different mobile devices enter and leave the joint presentation. The joint presentation adaptively molds itself to the ad-hoc collection of available devices.

Our focus on splitting a Web page to multiple devices shares some commonality with work on multi-device user interfaces. In one multi-device approach, a user can "pick" an object from a PDA and "drop" it on a PC screen or digital whiteboard [Rekimoto98]. Our work differs from this approach in our Web-centric focus, as well our ability to enable a PDA to simultaneously control multiple endpoints or output devices. Other work has focused on partitioning of a user interface between a PDA and one other device [Robertson96].

In addition to splitting the output of a Web page among multiple different devices, our design enables the output to be split among multiple different human users. As a complement to multi-device Web browsing, we have extended the design of WebSplitter to incorporate the notion of multi-user Web browsing. Our design goal is to provide functionality that will, for example, enable a lecturer to split a Web slide show to different users in an audience. This technique applies not only to a teacher presenting a Web-based lecture to multiple students, but also in general to other types of Web-based presentations, e.g. a CFO presenting a quarterly financial report to a group of stock analysts on a conference call. Integrated multi-user multi-device Web browsing is illustrated in Figure 1.

In addition to collaboration and multi-device support, another design goal 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. The ability to present 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 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.

The notion of joint Web browsing with multiple users is familiar to the field of collaborative or conferencing groupware, also known as computer support for cooperative work (CSCW). There are commercial realizations of collaborative Web browsing such as WebEx.com [WebEx] and Netscape Conference [Netscape], as well as academic projects such as UNC's CobWeb [Stotts97]. These techniques reproduce the same Web page on each user's browser, and do not permit the notion of creating limited partial views of the same Web page based on access privileges, as explained below. More recently, the VIEW Media conferencing system creates in a manner similar to our approach partial views of the same hypertext document based on access rights of individual users [Yokota99]. Unlike our work, VIEW Media does not address the notion of splitting a Web document to multiple devices, and does not offer a policy file as a for annotating Web pages so that partial views of the same page can be created. The CAT (collaborative active textbooks) system allows multiple users to see different partial views of the same Web animation show [Brown96]. Unlike our work, the student must manually point at a specific Web page that supports animation in order to receive the partial view. If the instructor changes to a new Web page, then all students must manually change to the new Web page. In contrast, WebSplitter automatically delivers updates, e.g. new Web pages, to participants in a joint browsing session.

The WebSplitter design enables the author of a Web presentation to create partial views of a Web page 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 an ongoing collaborative Web 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. We provide more detail on design considerations regarding the policy file in Section 2.3.

WebSplitter integrates the concepts of service discovery and user management to provide a single abstraction supporting both the collaborative and multi-device themes of this work. Support for collaborative browsing requires that each user be able to first discover the WebSplitter application and ongoing browsing sessions, prior to collaboration. This is a function that service discovery can typically provide. Conversely, support for multi-device Web splitting requires the WebSplitter application be able to discover users and their associated output devices so that Web content can be split to the proper destinations. Associating devices with their owners is a function of user management. WebSplitter adopts the model that service discovery and user management operate as a single abstraction that provides the means for discovery and registration of proxies, sessions, and 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/user management 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. In principle, the application need not know the particular service discovery protocol used to satisfy discovery or registration, e.g. SLP (Service Location Protocol) [Veizades97], Sun's Jini [Waldo99], Bluetooth's Service Discovery Protocol [Bluetooth], Berkeley's Secure Service Discovery Service [Czerwinski99], Microsoft's Universal Plug and Play [UPNP], IBM's TSpaces [Wyckoff98], etc. Similarly, the application need not know whether the service discovery registry is concentrated in a single server or distributed among multiple servers. The WebSplitter application relies on service discovery to provide geographic scope and locality. These issues are further discussed in Section 3.

The WebSplitter architecture consists of three components: server-side annotation of an XML Web page with an XML metadata policy file that defines access privileges to XML tags within each XML Web page for different groups of users in a browsing session; a middleware proxy that splits Web content based on the identity of end users, the capability of output devices, and the access privileges granted in the policy file; and a client-side component, e.g. applet or other technology, for user login and reception of pushed browsing data.

In the following, we present a description of the architecture of WebSplitter as well as our implementation experience and progress. Next, we describe the architecture of our WebSplitter design. The collaborative multi-device Web browsing architecture is divided into three components. In Section 2, we describe the server-side component, namely annotation of the XML Web content with an XML metadata policy file that defines privilege groups and default mapping rules of XML tags to output devices. In Section 3, we discuss the middleware proxy component whose responsibility is to split XML Web content based on the information contained in the policy file, the identity of the end user, and the various devices owned by the end user. In Section 4, we discuss the client-side component, either an applet or other client-side code capable of receiving pushed browsing data from the WebSplitter proxy. In Section 5, we discuss the progress of our implementation, and our experience with the feasibility of our Web splitting ideas. Section 6 discusses in more detail the relation of our WebSplitter design with prior work in collaborative conferencing as well as multiple small-device computing, also known as pervasive or ubiquitous computing [Want95].

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

2.1 XML versus HTML

The documents to be accessed and split into independent components during a Web browsing session are hypermedia documents. Specifically, we have chosen to rely on XML as the authoring language for the Web content. The Web community is aware of the limitations of HTML to develop rich Web content. An effort is underway to transition to XML as a markup language. W3 propositions like XHTML present a way to define compatibility between HTML and XML. Many Web servers already store their Web content as XML documents but convert them into HTML before sending them to the end user because only Microsoft IE5 can currently understand (a Microsoft version of) XML. But as more client browsers become compatible with XML, XML will become ubiquitous.

XML is highly appealing due to its flexibility. Every author defines his own tags, his own components. This allows WebSplitter to split at the granularity of tags the XML document into independent components 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. HTML doesn't allow the user to associate any meaning with the components of the Web page. In contrast, XML permits separate labeling of the two different tags for the navigation bar and for the slide. XML allows more flexibility in separating the different components. We'll see that some splitting can be done with HTML too but that it doesn't allow most of the features that the multi-device Web browsing proxy intends to provide.

2.2 The XML document

The full capabilities of the multi-device Web browsing proxy will only be exercised when the Web page is written in XML. 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. Second, an XSL document is also needed by an XML-enabled browser to transform the customized XML into a language that a browser can render. We assume that this XSL document will define the transformation from XML to HTML or another markup language; this ensures that even if our proxy is not used between the Web server and the client browser, a non-XML-enabled browser would be able to process the Web document.

Consider the following example of the XML Web page for a slide presentation as shown in Figure 2. A lecturer has written a Web presentation that is stored on a Web server and desires to show this presentation to other users, perhaps in a telephone

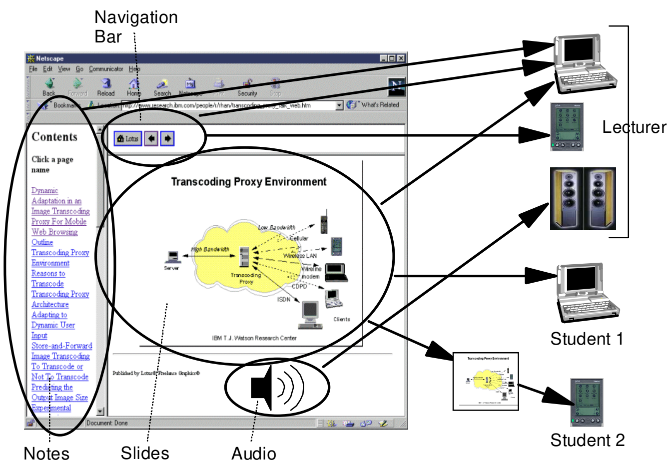

Figure 2. XML tags/components in a Web document (e.g. notes, navigation buttons, graphical slide, and audio) are separated and delivered to different users, according to user preferences and access privileges, and then are further subdivided for delivery to different devices owned by a single user (e.g.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

conference, conferencing room, or lecture hall. The lecturer may arrive with a WinCE device and a laptop. The lecturer may desire to navigate the presentation through a WinCE device, in order to have freedom to roam during the presentation. In this case, the navigation bar with forward/back buttons should be sent to the WinCE device. Simultaneously, the lecturer may wish 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. In addition, the lecturer may desire that the audio portion of the presentation be sent to the sound system available in the room. Also, students in the room may have PDA's and others may have laptops, as shown on Figure 2. The audience could be sent the slide (a transcoded version could be sent to 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 that defines how the above XML should be converted into HTML.

2.3 The policy file:

2.3.1 Adaptation to privilege groups and device capabilities

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 subscribers to a session.

```
<?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>).

The key responsibility of the policy file is the definition of privilege groups. The idea behind privilege groups is to provide a mechanism for differentiating the access privileges of participants in a collaborative browsing session. A session can offer differing *partial views* of a Web browsing session to different privilege groups. The author of a Web page that is to be presented to multiple users in a browsing session may wish to give some user groups higher or lower privileges to access certain XML tags than other groups. In Figure 4, the policy file permits a lecturer who is presenting a college class on-line to students via our collaborative proxy to limit 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).

We note that though a user may have the privilege to access a specific XML tag, the user may not have an output device capable of processing that XML tag. That's why the policy file should contain information on which tags can be seen as a function not only of the group the user belongs to but also as a function of the device the user has to take part to the collaborative session. But even with this restriction, we'll see later that the user can use several devices to compose a joint presentation (at least the whole view he is allowed to see.).

```
# Policy file for XML Web page of Figure 3

# Define groups in a collaborative session and passwords
associated with each group
group = "lecturer", password = lecturer_password
         Input_permitted = TRUE
group = "students", password = students_password,
         Input_permitted = FALSE

# Define the permitted set of tags that each group may
#  receive;  the permitted set of tags can be further divided
#  into subsets based on browser type or device capabilities

If (group = "lecturer")
    If access = "Internet Explorer" or "Netscape"
        Tags = presentation, head, title, nav_bar, slide,
               picture
        Undefined_Tags = images, text, audio, video
    Else if access = "Pocket Explorer" (WinCE)
            or "ProxiWeb" (Palm)
        Tags = presentation, head, title, nav_bar, notes
        Undefined_Tags = text

If (group = "students")
    If access = "Internet Explorer" or "Netscape"
        Tags = presentation, slide, picture
        Undefined_Tags = images, text, audio, video
    Else if access = "Pocket Explorer" (WinCE)
            or "ProxiWeb" (Palm)
        Tags = presentation, slide, picture
            (transcoding param's (a=_, b=_, ....))
        Undefined_Tags = text

# Define the media type of each tag name so end user
# understands purpose of each tag name
Images: nav_bar, slide, picture
Text: title, notes
```

Figure 4: Pseudo code for policy file.

2.3.2 The policy file syntax:

Our design leverages XML to define the syntax for the policy file, rather than inventing its own language for specifying the rules in the policy file. We find XML to be adequate for our needs of defining privilege groups, of listing the tags permitted to be received by each privilege group, and of expressing various other properties associated with each privilege group. Relying on XML also simplifies the composition task of the author, who doesn’t have to learn another syntax. In addition, relying on XML allows us to leverage off of the existing XML parser already needed to process Web pages; we can avoid having to write and integrate a second parser into the proxy.

Our design also views the policy file as a second file separate from the XML Web page that references the policy file. We wish to allow multiple Web pages to reference the same policy file. For example, this may be helpful to an author composing a Web slide presentation who wishes to define the same set of privilege groups over all Web pages, but does not wish to rewrite a new policy file for each page. Instead, the author writes a single policy file that is referenced by each page in the presentation. All pages referencing the same policy file share the same privilege groups. The author aggregates all XML tags that will be encountered while traversing the multiple pages in the presentation into the single policy file. An alternative to a separate policy file would be to embed the policy file’s information into each XML page. However, separating the policy file’s metadata information into a unique file makes it easier for the author to update privilege groups; the author simply changes the one policy file that is referenced by each Web page, without having to edit each Web page containing the same embedded definition of the same privilege groups.

Let us now consider an example policy file based on the pseudo-code of Figure 4. The policy file would be structured as an XML document. Within this document, we need to define our own namespace 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). The XML policy file corresponding to the Figure 4’s pseudo code is shown in Figure 5.

```
<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:undefinedta
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 5: XML policy file corresponding to the pseudo code stated in Figure 4.

A schema needs to be defined standardizing the tags and attributes the author should use in the policy file. We also need to standardize the value used for some of the attributes. For example, it is important that all authors use the standard tag name “group” in order that every collaborative multi-device proxy 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 so that the proxy can universally understand what is meant by the author, 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, etc. 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 special requirements regarding the capabilities of the device. For example, the author could write *<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 way to describe the capabilities, we cannot take advantage of this information. The devices need to use the same vocabulary as the author to list their capabilities with service discovery. The W3C draft "Composite Capability/Preference Profiles (CC/PP)" has followed this approach of globally standardizing the characteristics of devices but is facing difficulties finding support for its proposal [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 would have the ability to 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.

2.3.3 Subsequent policy files

The policy file is used to establish the rules governing which device is allowed to receive which subset of tags. During a Web browsing session, a user may encounter two different Web pages that reference different policy files, or even no policy file at all. If the most recent policy file takes precedence, then new privilege groups and new tag mapping rules would have to be established. To avoid this inconvenience, our design assumes that the initial policy file associated with the first XML page downloaded by the session into the proxy defines the privilege groups and hence tag mapping rules observed throughout the session. Though other pages with different policy files may be encountered, the privilege groups and tag mapping rules defined by the initial policy file overrule any subsequent policy files, as long as the user indicates that this is desirable. During session establishment, our design gives the session creator the option to suppress prompting of a user whenever a new policy file is encountered during the Web session. If the user specifies notification, then the proxy will pop up a window on the session initiator device querying the user about whether a new session should be established. Otherwise, queries will be suppressed and any new policy files encountered will be overruled by the initial policy file. If the initial Web page lacks a reference to a policy file, then the proxy defaults to a set of heuristic tag mapping rules based on the MIME/media type of tags.

In the case where the initial policy file does not match the current page's policy file, then undefined XML 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 admittedly coarse-grained, but the proxy has little option other than to fall back to this default form of page splitting. This undefined_tag mechanism is also effective for dealing with HTML documents. When an HTML document is accessed, then the default splitting rules will be taken from media-based partitioning expressed by the tag *undefined_tag*.

3. MULTI-DEVICE WEB BROWSING PROXY

The second major component of the WebSplitter architecture is the middleware proxy.

3.1 Placement of the splitting function in a proxy

Our design places the document-splitting function 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. In particular, the proxy establishes rules for the distribution of XML tags to each device subscribing to the session. The proxy bases its distribution or demultiplexing rules on the initial policy file, as well as the identity of the user and the capabilities of the user's devices. This mapping of permitted tags to permitted devices is reflected in Figure 9, which is placed in a later section that discusses how the end user may customize these mapping rules.

Placing the splitting function logically in a proxy provides some advantages. 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. Operating the WebSplitter proxy in personal server mode enables an ad-hoc network of personal devices to collaborate in the context of a Web browsing session. The proxy also has the capability to limit the processing that has to be conducted at the client end. This is especially desirable when the end-device is small and of limited capability. The proxy assumes most of the complexity of the WebSplitter application, so that devices like WinCE or Palm only need to run a small applet-like program to participate in a joint browsing session. The proxy can also perform prefetching of files: it will behave like a browser meaning that it will fetch the additional files needed (images, audio files…) without waiting for an end-user to request them. This improves efficiency. Proxies are also quite capable of traditional adaptation functions such as transcoding of images.

3.2 Control information exchange via HTTP

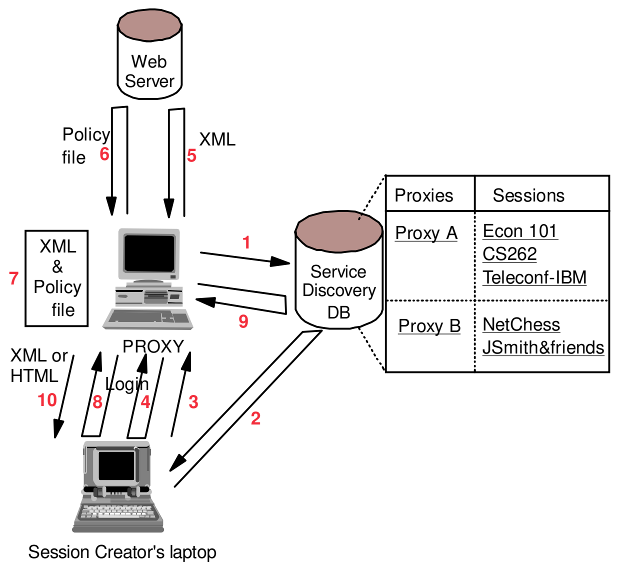

Figure 6: Procedural steps for establishing a session of collaborative multi-device Web browsing.

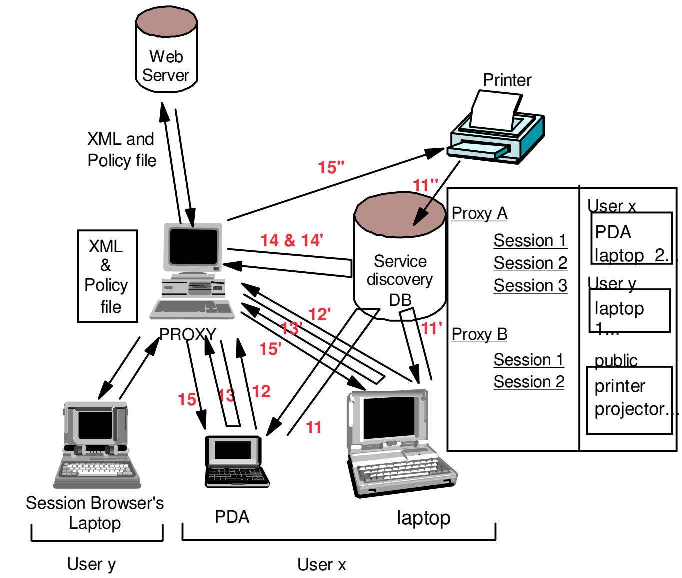

Figure 7: Procedural steps for joining an existing collaborative multi-device Web browsing session.

We have chosen to rely on HTTP exchanges for all the interactions between the proxy and browser-enabled end-users, rather than invent a new control protocol. There are two main reasons for this choice. First, the goal of our design is to leverage existing technologies as much as possible. Our desire is to use the standard Web browser interface and the standard HTTP protocol associated with it. In this way, we limit the modifications that have to be made at the client's side. The protocol used to transfer information between the proxy and the end users (or at least the Web-enabled ones) is standard HTTP, so nothing new has to be deployed. Second, this approach also has the advantage of being easy to use for the end-user: he just needs to fill Web forms like he is already used of doing.

For non-browser devices like speakers or wall displays, the interaction will use the specific protocol requested by the end-device. We expect service discovery to contain an attribute that would specify the access mode to the end-device. The proxy uses this method to send the components this device is supposed to receive.

3.3 Establishing a joint browsing session

Figure 6 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 part of the requested XML page that the session creator is permitted to receive is returned to the session creator (10).

3.4 Subscribing to an ongoing browsing session

Figure 7 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 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 appropriate end clients/users (15, 15', 15'').

3.5 The role of service discovery

As shown in Figures 6 and 7, service discovery plays a supporting role in the design of the WebSplitter application. In collaborative multi-device Web browsing, both users as well as devices must have some means of registering their services as well as discovering services that have been registered by others. Some service discovery mechanism is needed so that the multi-device Web browsing proxy can advertise itself and its ongoing sessions to potential users. Earlier, we listed a variety of service discovery protocols. The mechanism for registration and discovery of associating users with devices requires a user management service. We look at service discovery/user management as a black box that is able to support a simple API consisting of register() and query() functions. Within this black box, it is left up to service discovery/user management to determine whether its registry, which we conceptually refer to as a service discovery database (SDDB), is centralized or distributed. Wide area support will influence whether our proxy is able to support distance learning with remote students. 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 collaborative multi-device function offered by the proxy.

The first task of the proxy is to advertise itself by registering its name and its IP network address via a service discovery protocol to the SDDB registry. The proxy also specifies a URL that the user should access to receive a session establishment form. A user querying the list of available services starts a multi-device session by pointing and clicking to select the proxy. In response, the user receives the session establishment menu from the proxy.

The proxy also registers its public ongoing sessions with service discovery to allow users to join a session. The proxy specifies a URL associated with each session. This URL points to a login form dynamically built by the proxy for this session. In the same way a user selects a proxy, a user clicks on the session name, which takes the user to the login form discussed below.

3.6 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 the policy file from the Web server. This initial policy file defines privilege groups that will be used throughout the session. The policy file specifies what subset of XML tags each privilege group is permitted to receive. The policy file may provide further detail that enables tags to be demultiplexed with finer granularity within each privilege group according to the type of access device. Refer to Section 2.3 for more detail regarding the policy file itself. The proxy uses the policy file to establish general rules mapping XML tags to privilege groups.

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. Combining those two sources of information, the proxy will build dynamic rules that map permitted XML tags to specific users.

As a final refinement of the tag mapping rules, the proxy needs to map XML tags to the multiple devices owned by 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.

In case the first Web page lacks a policy file, i.e. the requested file is written in HTML or is written in XML but doesn't conform to WebSplitter's architecture by failing to reference a policy file, the proxy will lack information about privilege groups, and will not know how to map Web content to users. The content mapping rules must fall back to some heuristic. In the absence of a policy file, the proxy creates a single default privilege group. Access to this single privilege group is open, since no password has been defined. Any user that joins this session is granted access and will only be able to split Web content among his devices based on media type. A set of heuristic rules mapping Web content to devices is created based on the end device's capability as well as media types within the Web content. A full non-split HTML or XML page will be sent to devices supporting a standard browser. Also, the appropriate media type will be sent to media-specific devices, e.g. audio to a stereo system.

3.7 Login of the users

Using the information found in the policy file, the proxy builds a customized login page for this session as shown in Figure 8. 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 needs to know the subscriber's username in order to associate the output devices registered under that username in the SDDB with the subscriber. The provided username should match the username that was registered by the devices belonging to the subscriber. If there is a difference in usernames, then the proxy will be unable to associate the devices with their proper owner. Once the username is known, the proxy retrieves the devices registered under that username and builds 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.

The password field is used by the proxy to identify the privilege group to which the user belongs. 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, such as an email from the person establishing the session.

We allow some privilege groups to be non-password protected, i.e. openly accessible. Since not all members of an audience may be known a priori, then prior password distribution cannot always be assumed. For example, some students who wish to subscribe to an ongoing browsing session may have forgotten their password, never received a password, or may be new to a class. In these instances, our support for public non-password-protected privilege groups enables such students to easily join an ongoing browsing session without requiring every subscriber to know the password. In these cases, the login menu not only displays a password field, but also a list of the non-password-protected public privilege groups. An example of this hybrid scenario is shown in Figure 8, in which the public privilege groups "lecturer" and "students" for the slide presentation are not password-protected. If a user wishes to join a public group, then the user simply checks the box of the desired public group. Otherwise, the user enters the appropriate password. Finally, in case there is no initial policy file associated with the first requested Web page, then the proxy creates only one public privilege group. Consequently, only one privilege group is listed with a checkbox, and no password field is shown. The password defined in the policy file is encrypted, as is the response in the login menu, to ensure security.

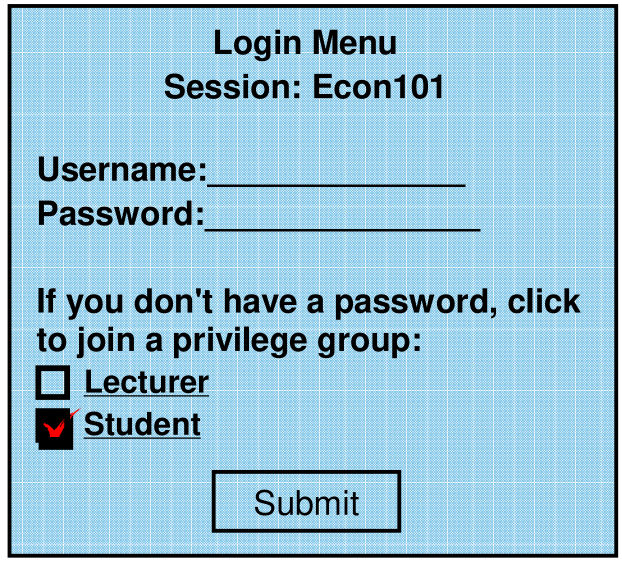

Figure 8: Login/registration page sent to a new subscriber's browser from the proxy.

3.8 Constructing the receiver's presentation

3.8.1 Creation of transformation files and partial views

Following user login and retrieval of the user's list of devices, the proxy will have sufficient information to construct mappings of XML Web content, as signified by XML tags, to a user's devices. When the target device is an XML browser, the tag mapping rules 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, in order to describe the layout of the sub-document or partial view on the user's browser window. Combined with the original Web document, this transformation file lays out the sub-document that has to be sent to the device used by the user. The format of this sub-document (for example XML or HTML) depends on the capabilities of the device. These capabilities can be learned from service discovery or in the case of browsers, with the User-Agent field of HTTP requests. Section 5.1 further discusses transformation files.

3.8.2 Multiple windows/control panels

For devices that support 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 9. This third window enables the user to customize how Web content is demultiplexed to the user's various devices.

For other devices like PDA's that don't support multiple windows, we need another method for supporting access to logout and personalization functions. Inserting these buttons into each page is an option. A second option is to embed these functions directly into a client-side software component that is installed on single-window devices. For example, as we'll see in the next section, another requirement of the client-side component is the ability to support data-push technology. This requires client-side code, e.g. an applet. Since many browsers on limited devices still don't support data push, e.g. non-enhanced Pocket IE on WinCE devices or HandWeb on Palms (though WML browsers should eventually support data push [WAP]), then it becomes necessary to require some custom client-side code to support data push protocols. Given that a client program needs to be written for such devices, we can integrate the functionalities provided by this control panel into such a program for execution on these devices.

4. CLIENT-SIDE FUNCTIONS

4.1 Data push model

Each time a session presenter requests a new page, this update must be pushed to all the browsers and devices, like 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 display 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 chunk of data, including a directive (in the HTTP response or the document header) that says "reload this data in 5 seconds", or "go load this other URL in 10 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. Since the continuous polling overhead of pulling clients can be quite inefficient, we have chosen a server-push approach.

Unfortunately, all the flavors of browsers don't have the same characteristics: Netscape and IE don't use the same techniques to implement server push or client pull. For example, to do server push (which allows a server to push data to a browser), Netscape relies on a specific MIME-type called "multipart/mixed", which lets a single message (or HTTP response) contain many data items. The different parts of the pushed data are separated by a defined delimiter string. But IE browsers don't use this MIME-type so we cannot use this method to push data to them. As for pocket IE used in WinCE devices, it doesn't implement the full HTTP specification or support Javascripts, which would allow us to use a client pull technique. Given the diversity of supported features, we need to have a different way to push the data for each browser type (at least for the most common ones).

Currently, data is pushed to end users via unicast sockets, rather than multicast channels. The data pushed to each user is quite heterogeneous in nature, varying due to the diversity of end devices as well as the customization preferences of the end user. Web content sent to a PDA will differ markedly from content sent to a full-fledged browser. In addition, the splitting rules for a given page can vary for each user. The lack of commonality of the split content pushed to different users negates one of the primary advantages of multicast, efficient delivery of common content. Multicast may still play a role in the future for video and audio.

4.1.1 Java-enabled browsers

Thanks to Java, there is an alternate 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 achieving the same effect via a Netscape plug-in for Netscape or a Microsoft Active X component for IE, we write one applet capable of implementing data-push on any Java-enabled browser.

4.1.2 Non-Java browsers

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 WML browsers intend to support data push, but this technology is still in the development stage. If we want to be able to include those devices in a Web browsing session, we have to loosen the goal we had fixed not to install anything at the client side. Otherwise, if we try to use the existing technology, there's nothing that allows us to push data at a PDA browser. So until the most popular PDA browser support java and applets, a user that wants to use our multi-device Web proxy from his PDA will have to install a program on his device.

For WinCE devices, we are implementing 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 will implement a data push functionality, receive Web content and display it on the WinCE device. For Palm pilot, we are investigating to see if there is existing software that we could extend to incorporate push technology.

4.1.3 Non-browser devices

Many non-browser devices, such as a stereo system or projection display, are inherently poised to listen for and process pushed data. For such devices, it is a matter for service discovery to specify how the devices register their capabilities with the SDDB registry. Service discovery also must reveal to the proxy the method for accessing non-browser devices, i.e. the data format and push protocol that the non-browser device is expecting. Another team at our institution is implementing the service discovery component of the WebSplitter architecture to enable devices to advertise their service, e.g. speakers available in the room advertise via a Bluetooth attachment.

4.2 Personalized configuration of the XML tag mapping rules and session parameters

After a subscriber has joined an established session, or a session creator has finished with session establishment, each participant in the session may configure the mappings of tags to devices according to the user's personal tastes. The proxy initially uses the information provided by the policy file to construct a default mapping of permitted tags to permitted output devices. However, this default mapping may not satisfy the end user. As mentioned in Section 3.7, this default mapping is exposed to the end user to allow the user to customize where Web components are sent and displayed. Conceptually, each tag is permitted to be sent to a list of permitted output devices, as shown in Figure 9. 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 the spirit of ad-hoc collaboration, new devices can be added to and removed from the list of permitted output devices in Figure 9. For browser-enabled devices, a customization applet is downloaded to implement the ability for the user to change the distribution of components among the devices.

We introduce a button in the control panel described in Section 3.8. Recall that, in the case that the end user has a Netscape/IE browser, then there are two screens received by the end user. The first screen contains a control panel, while the second screen contains the partial view of the Web page. The first screen will contain at least the following two buttons: a "personalize" button, and a “logout” button. We focus on the “personalize” button here.

When the user clicks on the “personalize” button, the proxy is contacted and returns a screen of information conceptually similar to Figure 9. The proxy has all the information needed to create the mapping of permitted multimedia tags to permitted output devices. The tags are taken from the XML file and their media type from the rules file. We need to specify the media type of each tag to give a better understanding to the user of the usage and meaning of each tag name. The user identity and group to which the user is associated is taken from user login/registration. The list of available devices and their owner are taken from user management/service discovery. We also expect that the devices would have registered their capabilities with service discovery so that the proxy knows if a device can play audio or display pictures. This allows the proxy to build a mapping of permitted tags to permitted devices.

In Figure 9, the list of tags permitted to be received by this user are shown on the left, while a list of output devices that the user has permission to send to are shown on the right. Conceptually, each individual user simply draws an arrow from tag to output device in order to send the multimedia represented by the tag for display/playback on the output device. In practice, rather than use graphical arrows, a nested hierarchy is shown in which each output device has a list of tags that are being sent to that device.

We want to give the possibility to the user to register a device after he has started to do the customization. If the device the user wants to use is not in the list of possible devices, it’s because the user hasn’t registered it with service discovery/user management so when the proxy queries service discovery/user management to get the list of devices that this user could use, this device was not present. The refresh button shown in Figure 9 signals the proxy to save the current associations and to make a new query to service discovery/user management to discover which devices are accessible to this user. This allows the user to register a device after joining an ongoing session without losing the customization choices already made.

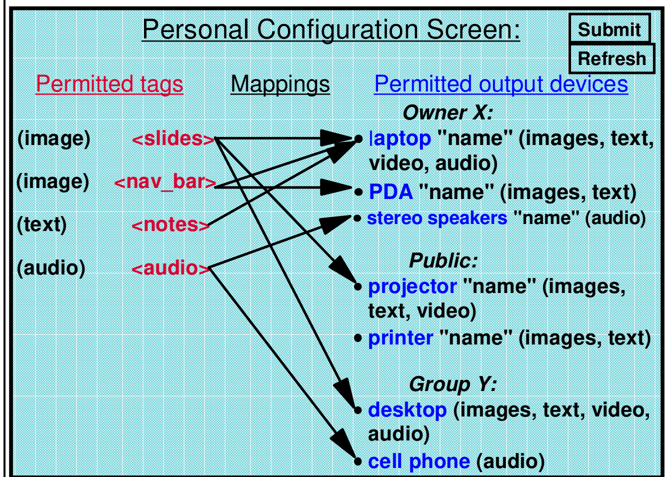

Figure 9: Conceptual model of a personalized configuration screen enabling the end user to customize the mappings of which tags are delivered to which end devices and which users. The arrows signifiy that a given tag is to be sent to a given device.

5. IMPLEMENTATION STATUS

Our initial prototype focused on browser-enabled clients. Our initial 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 6, as well as Steps 12, 13, and 15 in Figure 7. We will have more to say our progress towards implementing service discovery below.

5.1 Transformation files

After the user has logged in, the proxy knows which subset of the original document (which tags) can be received by the user (given the privilege group and the target device). So the proxy can build a transformation file as we mentioned in Section 3. The goal of this transformation file is not only to select the tags from the original XML document but also to create a document that the end-device can understand. In a standard Web browsing scenario, the document sent to the client is an HTML document but in the case of the multi-device Web browsing, the end-user can be very heterogeneous: it can be a standard Web browser like Netscape or IE, but it can also be a cell phone browser that understands 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 have chosen to use XSL for the transformation files customized for this user. XSL provides a very powerful way to select tags in the original XML. For each user, we build an XSL file that will select from the original XML file only the tags that the user is allowed and capable of receiving. The output is a new XML file which contains a subtree of the initial tree. We need to apply another transformation file to this subtree to produce a document in a format understandable by the intended end device. Again XSL seems to be appropriate for this transformation file, at least in the case when the output file should be a markup language like WML, HTML, XML or even for a pdf output file. For producing the data to be sent for example to a speaker, we won’t be able to use such a transformation file because a speaker could be expecting binary data. For these kinds of devices (speakers, displays), the proxy will have to use the access mode specified by those devices when they registered with service discovery. But for text output, XSL should permit to create the sub-documents. Ideally, the output documents should respect the style that had been chosen by the author. That’s why we should make use of the original XSL that was associated with the original XML document. If the format requested by the end device is HTML or XML, we could try to modify the original XSL so that it doesn’t reference the tags that are not permitted for this end-user. If we can do that, we can just apply this XSL to the subtree to create an HTML document. For other markup languages like WML, we will need an engine to convert HTML into WML. For the time

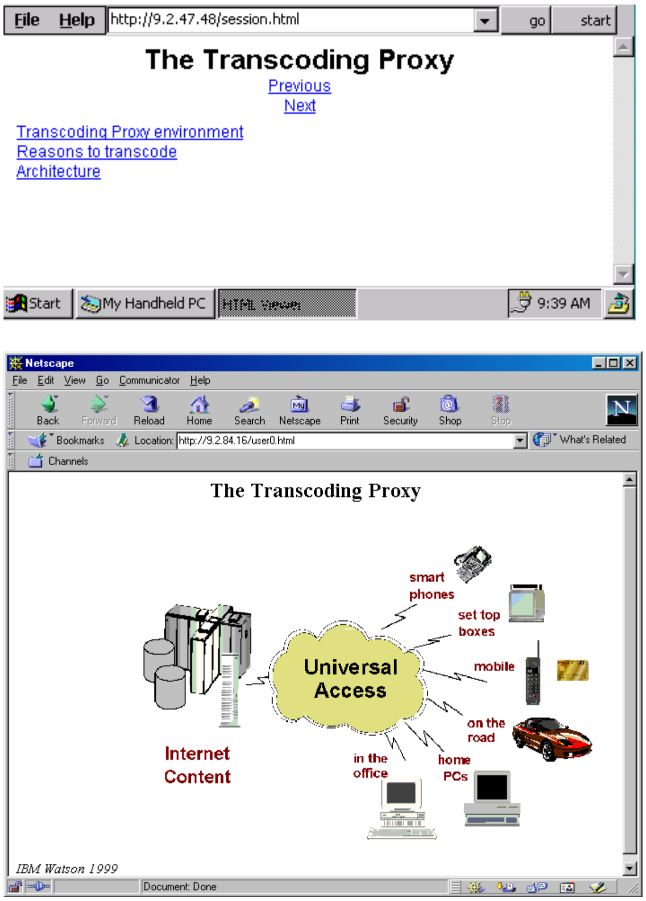

Figure 10: Screen captures of our current implementation of WebSplitter. The XML Web page of Figure 2 has been split by our proxy to present only the private text notes plus navigations buttons (transcoded into text) to the WinCE's screen (see top), while only the graphical slide appears on a typical student's browser. The private notes are hidden from the student. Also, the graphical slide is not sent to the WinCE user, due to the PDA's limitations.

being, we create dynamically the XSL that will allow us to subset the original XML document but we are still investigating how to create the second XSL transformation file to respect the style desired by the author.

5.2 Status of implementation

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 policy files. The default configuration for redirection of output is that the proxy sends all the components to the requesting machine (the ones it is allowed to and capable of receiving). We have defined multiple privilege groups by defining several groups in the policy file. Multiple users can log into the proxy and are identified as belonging to different privilege groups. The proxy employs different XSL files for different privilege groups and thereby sends different partial views of the initial document to member of different groups. We have successfully implemented the data-push protocol via an applet to enable joint browsing.

Our initial implementation of multi-device Web browsing does not rely on service discovery. The proxy does not register itself or its on-going sessions (Steps 1 and 9 in Figure 6). The users that want to establish a session have to know the URL for session establishment, thereby bypassing Step 2 in Figure 6. The users that want to join an on-going session have to know the URL of the session, thereby bypassing Step 11 in Figure 7. Moreover, the prototype proxy does not attempt to dynamically discover the devices available in the room (bypass Step 9 in Figure 6 and Step 14 in Figure 7).

Using this manual method of discovering and logging into a WebSplitter proxy, we have been able to demonstrate the splitting of a Web document to two different users as well as to two different 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 10 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 user of the WinCE device essentially receives just as much information as is needed to perform remote control of the presentation. The lower picture shows what a typical user receives on his browser, namely the graphical slide only. Consequently, the policy file has successfully filtered out the Web objects to which this user doesn't have access.

5.3 Future work

Integration of service discovery and user management into our current implementation of WebSplitter is a key focus of our to-do plans. Another team at our institution is in the process of deploying the service discovery and user management component of the WebSplitter architecture. Requiring users to know which URL to access and hard-coding the list of devices allowed us to prove the feasibility of breaking a document into components and of selectively sending a combination of those components to different devices. Our next step is to enable users to dynamically find the multi-device Web browsing service and for devices to dynamically collaborate with the proxy in a manner that is transparently painless to the users.

Service discovery will allow us to discover the access mode for the devices and the format of data they expect. For browser enabled devices, we can use IP for access mode and the User-Agent field is sufficient to determine what format to use to send the data; but for non browser enabled devices, service discovery should provide this information to the proxy. The XSL transformation files are very suitable to create markup documents like HTML or XML so they should be used to create the sub-documents going to browser enabled devices. For the non-browser devices, the proxy will have to use the information advertised by service discovery to know how to format and push the data.

6. RELATED WORK

The notion of joint Web browsing with multiple users is familiar to the field of collaborative groupware, also known as computer support for cooperative work (CSCW). There are commercial realizations of collaborative Web browsing such as Netscape Conference [Netscape], as well as academic projects such as UNC's CobWeb [Stotts97]. Most of these techniques reproduce the same Web page on each user's browser, and do not permit the notion of partial views of the same Web page. RealNetworks' Real Presenter allows multiple recipients to view a PowerPoint presentation on the Web [Reala] for which the presenter has recorded a narration or video with each slide. In the live broadcasting version, the audio and slides data are sent to a RealServer which then streams the data to any user who clicked on the link of the presentation [Realb]. Microsoft's NetMeeting [Microsoft] allows participants to collaborate and share Windows documents in real time over the Internet. All the users see the same information on their screen. There are other products with similar features, e.g. ICP from VocalTec or Netscape Conference [Netscape]. Netscape Conference has a collaborative browsing feature that lets two people see the same Web pages at the same time via the Internet. WebEx.com enables multiple users to follow the same presentation online [WebEx]. The screen of the browsing user is captured, sent to WebEx.com's server, where it is then relayed to the other users in the conferencing session. There is support not only for Web browsing but also for joint display of documents from any application (e.g. Microsoft Excel documents or pdf documents...). Virtual Network Computing (VNC) uses a Java-based architecture to similarly share the results of screen scraping [Richardson98]. Also, a project at the University of North Carolina built a collaborative Web browsing system [Stotts97]. The goal was to allow multiple individuals (a leader and followers) on separate geographically distributed workstations to collaboratively browse the Web. The participants can see the Web pages fetched by the leader automatically. The method the interaction between the leader and the audience is determined by a collaboration protocol chosen at the beginning of the session.

More recently, an idea similar to our own has been proposed that allows different users in a joint Web browsing session to receive different partial views of the same Web page [Yokota99]. VIEW Media is a cooperative hypermedia system that allows a conferencing system to limit the views that different users could have to shared documents, based on the access rights of each user to the shared documents. VIEW Media, similar to our work, views a hypermedia document as consisting of multiple components. Different subsets of these components can be selectively sent to different recipients of a groupware session. Unlike our work, VIEW Media is not based on XML, and appears to hide data by overwriting rather than split Web content in order to achieve partial views. Further, the authors' implementation appears to differ from our own in that there is no mention of the equivalent of a policy file as we have defined as a means to annotate Web pages so that partial views of the same page can be created. Finally, the authors do not address the notion of splitting a Web document to multiple devices.

In other related work, the CAT (collaborative active textbooks) system allows multiple users to see different partial views of the same Web animation show [Brown96]. An instructor's screen may contain multiple animation views of the same algorithm, along with a control screen to adjust the animation (e.g. speed). Each student points at the same Web page and receives multiple views of the animation, but not the control screen. In this way, the instructor controls the pace of the animation, but the student controls what views the student sees. Each student can interact with their own views, independent of the instructor and other students. Unlike our work, the student must manually point at a given Web page to receive the partial view. If the instructor changes Web pages, then all students must manually change to the new Web page. In contrast, WebSplitter automatically delivers updates, e.g. new Web pages, to participants in a joint browsing session.

Shared remote control of a video conferencing application was proposed in [Hodes99a]. A PDA is used as a remote control for manipulating a video conferencing application, rvic (remote video conferencing). This application of remote control deals with streaming conferencing application, not the Web. There is no notion of splitting output to multiple devices, nor is there a notion of limiting what other viewers can see.

Stanford's Interactive Workspace Project has initiated experiments with multi-device, multi-user environments [IWork]. This project investigates how to interact with the room infrastructure and involves sensors to get location information. Processing can be moved from one computing device to another to support and facilitate group interactions. The user would interact with multiple devices. For example, as the user moves in the room, the device he is interacting with to see his application or complete a task would change. Our understanding is that Web documents are not split in the manner we have proposed.

7. SUMMARY

We have proposed an architecture and detailed a design that enables multiple devices and multiple users to collaborate and jointly share the output of a Web browsing session. The WebSplitter design allows dynamic creation of groups of machines that will work together to provide a richer dimension to the Web browsing application. A single user owning multiple output devices, such as a laptop, PDA, and stereo system, may take advantage of our proxy to redirect different media objects within each page in a Web browsing session to different output devices. When there are multiple users, then the proxy enables different participants within a session to have different partial views of the same browsing session. This can be helpful in a presentation when the session presenter wishes to hide personal reminder notes from the audience. The proxy expects Web sites that wish to present their XML content to multiple users to have authored an XML metadata policy file that defines multiple privilege groups. Each privilege group has access to a different set of XML tags. Each user logs into a joint browsing session and is identified as a member of a given group. The proxy then requests from service discovery all of the devices that the user owns. Given all of this information, the proxy can construct an overall mapping of permitted tags to permitted output devices, i.e. each tag that the user is permitted to access is mapped to one or more output devices that the user owns or is permitted to access. If this proxy runs on clients' machines, users could spontaneously use the surrounding infrastructure or start a collaboration with other users anywhere they go. Our work combines four major themes: Web browsing, collaboration, heterogeneous devices with wired and wireless connectivity, and service discovery.

8. REFERENCES

[Bluetooth] Bluetooth Service Discovery, *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.

[Czerwinski99] S. Czerwinski, B. Zhao, T. Hodes, A. Joseph, R. Katz, "An Architecture for a Secure Service Discovery Service," *ACM/IEEE International Conference on Mobile Computing and Networking,* 1999, pp. 24-35.

[Esler99] M. Esler, J. Hightower, T. Anderson, G. Borriello, "Next Century Challenges: Data-Centric Networking for Invisible Computing, The Portolano Project at the University of Washington," *ACM/IEEE International Conference on Mobile Computing and Networking*, 1999, pp. 256-262.

[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*

[NetscapePush] Netscape PushPull, *http://www1.netscape.com/assist/net_sites/pushpull.HTML*

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

[Realb] *http://www.geocities.com/SiliconValley/Pines/7812/real.htm (require realPlayer)*

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

[Richardson98] T. Richardson, Q. Stafford-Fraser, K. Wood, A. Hopper, "Virtual Network Computing," *IEEE Internet Computing*, vol. 2, no. 1, Jan/Feb 1998, pp. 33-38.

[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.

[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*

[Wyckoff98] P. Wyckoff, S. McLaughry, T. Lehman, D. Ford, "T Spaces", *IBM Systems Journal*, 37(3), August 1998, pp. 454-474.

[XHTML] *http://www.w3.org/TR/xHTML1/*

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

[XSL] *http://www.w3.org/Style/XSL/*

[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.