Browser State Repository Service

Henry Song, Hao-hua Chu, Nayeem Islam, Shoji Kurakake, and Masaji Katagiri

DoCoMo Communications Laboratories USA, Inc.
181 Metro Drive, Suite 300, San Jose, CA 95110 USA
{csyus, haochu, nayeem, Kurakake, katagiri}@docomolabs-usa.com

Abstract. We introduce *browser state repository (BSR) service* that allows a user to save and restore multiple independent *snapshots* of web sessions on a browser. At a later time, the user can retrieve any saved snapshot on a potentially different browser on a different device to continue any one of the chosen saved session in any order. The web session snapshot captures a complete browser running state, including the last page that appears on the browser, document object state, script state, values that a user enters in forms on the last page, browser history for back and forward pages, and cookies. BSR service consists of a *browser plug-in* that takes browser session snapshots, and a *repository server* that stores snapshots securely for each user. The main contribution of BSR service is *that it decouples association between browser state and a device*, in favor of *association between browser state and its user*.

1 Introduction

Most of the web applications today are *session-oriented*. Most websites require a client browser to first establish a web session and obtain a session ID. This session ID can then be used by a website to track and identify the client browser as it moves between different web pages within the website domain. During a web session, a client browser may accumulate *session state* (e.g., session identifier, shopping cart, form inputs, and etc.) that is needed to interact with the web application servers over the stateless HTTP protocol. The session state on a browser can appear in cookies, document objects, and script objects (e.g., JavaScript or VBScript). When the client browser exits a website, the active web session is closed and some of client-side session state is un-recoverable.

This session-oriented model places a limitation such that *for the duration of a web session*, a user cannot switch devices; otherwise, she might lose her active web session and would need to restart it on a new device. Consider a case when she is running an active web session on a stationary device (a desktop PC), but an alternative mobile device (a Pocket PC with wireless access) is available to her. She would like to continue browsing but on a mobile device. However, she could not do so without losing her active web session on the stationary device. She is restricted to the location of her stationary device. Consider another case when she is running an active web session on a mobile device (a Pocket PC) with a small screen, but an alternative stationary device (a desktop PC) with a large screen is available to her. She would like to use the large screen of the stationary device, but again she could not do so without losing her active web session on the mobile device. She is tied to the device that she starts a web session with. To address these limitations, we propose a *browser state repository (BSR) service* that enables her to seamlessly or effortlessly migrate an active web session to any device that is accessible or convenient to her. A unique aspect of the BSR service is that it is based on a *client-side approach* that has many advantages over existing server-based approach that can also address this limitation. These advantages are described in details in section 1.3.

1.1 Motivation Scenario

We will use the following scenario to further illustrate motivations of our work:

> *Jane was shopping for window draperies online. Jane started by browsing an online drapery store using her office PC while she was in her office. When she was asked about sizes of her windows, Jane did not know them, so she stopped browsing the online drapery store. After work, Jane went home to get measurements on the sizes of windows. She returned to the online drapery store using her home PC, and continued shopping for her window draperies. After making her selections, she was not sure if the online drapery store offered good prices, so she visited a local store carrying her Pocket PC (or PDA) with wireless network access. On her way to the drapery aisle, she saw some beautiful wallpaper that fits perfectly with her selected drapery. Unfortunately, the local store did not carry enough stock. She used her Pocket PC to do an online search for the home page of the company that produces this wallpaper. Then she continued to the drapery aisle. She compared the prices of the online drapery store with prices found at the local store. After she checked that the online prices were good bargains, she ordered draperies from the online drapery store using her Pocket PC. When she got home, she used her home PC to retrieve the homepage of the wallpaper company. The homepage provided her with contact information, and she placed her order over the phone.*

We make two observations in Jane's scenario.

- *Multiple Devices*: Jane had multiple computing devices (office PC, home PC, and Pocket PC), and she used different devices to perform her online task(s) at different locations.
- *Multiple Tasks*: Jane had multiple ongoing tasks (ordering drapery, and finding contact information of wallpaper company), and she performed them in *discontinuously incremental steps* rather than one continuous step from the start to the finish.

Jane would like *flexibility* to use whatever device is accessible or convenient to her to continue her unfinished online tasks in an effortless or seamless manner. In Jane's scenario, she returned to the online drapery store on her home PC, and she would like to continue exactly where she left off on her office PC without having to start over again. To provide this flexibility, it requires the decoupling of an association between browser state to a device, in favor of an association between a browser state and its user and independent of any device. The BSR service is designed based on this new association. It enables Jane to seamlessly migrate her browser state with her across different devices. The mechanism used in BSR service is simple but effective. Before Jane switches out of her current device, she preserves her active web session by saving a snapshot of her current browser state on a trusted BSR repository server. When she finds a new device at a later time, Jane retrieves the browser state snapshot from the BSR repository server and restores it on the new device. Then Jane can continue with her online activity on the new device. The session migration becomes seamless and effortless for her.

An additional benefit of using BSR service is that it helps people to keep track of their ongoing tasks and do them incrementally in any order convenient to them. In Jane's scenario, she switched from one online task (ordering drapery) to another online task (finding contact information of wallpaper company). Using BSR service, Jane could juggle between several online tasks. Each online task-in-progress would be preserved in a browser state snapshot on a BSR repository server, and Jane could freely stop and continue any task at any time from any device.

1.2 Client-Based Approach vs. Server-Based Approach

To accommodate users like Jane who are mobile and have a need to do tasks on different devices at different locations, a service like BSR is needed to allow Jane to save and retrieve intermediate session state. There are two possible approaches in the design for this repository service called *client-based approach* and *server-based approach*. BSR service chooses the *client-based* approach that captures session state through a browser state snapshot, and stores this snapshot on a repository server (not on the web application server). In the client-based approach, web application servers can be made completely unaware of a session migration occurring on the client browsers and devices. The reason is that *HTTP protocol is stateless*. A client browser on a different device can connect to a web application server to continue an ongoing web session, as long as the client browser can present the same browser state (e.g., cookies, hidden forms, and etc.) to the web application server.

In the server-based approach, intermediate session state is captured and saved on the web application servers. It is used by many of advanced websites such as amazon.com or expedia.com. These websites typically require Jane to first register with them to obtain unique sign-on names and passwords. By identifying her sign-on name, websites can track Jane on different devices and retrieve any saved intermediate session state for Jane. However, there are limitations in server-based approach when comparing to client-based approach:

- *Incomplete Browser State*: the session state that the web application server can save is limited to information that a client browser sends to the web application server, which is unlikely to be the complete browser state. For example, it cannot capture browser state such as the last web page (if retrieved from browser cache) that Jane navigates to, values of document objects on the last web page, values of scripting objects, values of forms that Jane filled but had yet submitted to the website, and the browser history for back pages and forward pages. Consider the amount of efforts needed for Jane to switch from a PC to a Pocket PC using the server-based approach. She would need to re-construct the browser state on Pocket PC browser by signing-on with the online drapery store, navigating her Pocket PC browser to the page where she left off on the PC browser, and re-entering any empty forms. This can be considerable effort. In contrast, the client-based approach can capture and restore complete client browser state, making session migration across devices transparent and effortless.

- *Privacy*: Jane may not trust websites to keep track of her intermediate session state, which may involve personal information that she does not want to share with websites before she makes her final purchase (e.g., the website may intentionally increase the price of items in her shopping cart). She would prefer a trusted third party, such as a BSR service provider, to maintain her intermediate session state.
- *Decentralized Storage*: In server-side approach, each web application server maintains its own intermediate session state with Jane. If Jane has multiple incomplete online activities on multiple websites, Jane needs to remember all these websites. This can be a problem when the number of such websites becomes large. In comparison, the client-based approach such as BSR service uses a centralized repository server to store all her incomplete online activities. Jane would only need to remember the location of the centralized repository server to retrieve all saved sessions.

In addition to the limitations described above, there is one more reason that the client-based approach is better than the server-based approach. That is only small portions of web websites are advanced like amazon.com and expedia.com that allow users to preserve intermediate session state on the web application servers. Some websites do not identify and track users, because they do not want to burden users with the registration and sign-on processes until they are ready to make purchases. These websites will not save any intermediate session state for Jane. This means that after Jane closed the session with the website, she would lose her shopping cart and would need to start over again on a new device. Some websites store intermediate session state on browser-side cookies. Since cookies are not access-able to browsers on other devices, Jane would again need to start over again on a new device. On the other hand, the client-based approach is applicable to any websites regardless of whether or how they perverse intermediate session state for users.

There can be a possible redundancy in a snapshot if the BSR service is used in conjunction with a website that keeps track of the session state using the server-based approach. That is, a portion of the snapshot may be redundant information that a website also maintains, and that portion of snapshot can be retrieved directly from the website rather than from the snapshot. BSR service must be properly used to ensure that the session state stored the snapshot is consistent with the session state on the website. After a user takes a snapshot of a browser session, the user should not continue with browser session because further actions can create inconsistency between the session snapshot and the data on website.

We identify several design requirements for BSR service so that it can be easily deployed to current WWW. BSR service should require little or no modifications to the existing web application servers to support, and it should require little or no modifications to browser internal. A key assumption in BSR service is that websites do not set short time-out policies that automatically close a session-in-progress with a client browser after the session-in-progress is preserved and becomes inactive.

The rest of the paper is organized as follows: Section 2 discusses related work; Section 3 describes design of BSR service; Section 4 explains its implementation; and Section 5 draws conclusion and future work.

2 Related Work

BSR service is similar to bookmark concept on existing web browsers (it is also called "Favorites" on Microsoft Internet Explorer). Bookmarks allow a user to save URLs to web pages, so that she can quickly come back to these pages at a later time. An alternative solution to BSR service is to simply synchronize bookmarks across devices, but this solution is insufficient. Bookmarks can only work on static web pages which do not accumulate any runtime browser state with users, e.g., these static web pages do not contain client-side scripting and cookies to track web sessions. In comparison, BSR service can work on both static and dynamic web pages. As a matter of fact, most web services available today on Internet make extensive use of browsers' capabilities to support client-side scripting and cookies to track sessions.

Application-layer mobility in SIP [1] shares a similar goal with BSR service. SIP provides session mobility that allows a user to change terminals while maintaining the same running media session. It provides personal mobility that allows other people to address a single user located at different terminals. It also provides service mobility that allows a user to change terminals while maintaining access to services. However, SIP is targeted toward telecommunication services, such as video conferencing, voice over IP, and instant messages, whereas BSR service is targeted specifically to provide session mobility for web applications.

Service hand-off [3] in the Iceberg project and Universal Inbox [4] describe architectures that can support personal mobility and service mobility for a user who may want to switch between heterogeneous access networks or between heterogeneous access devices. They deal with issues such as data transformation into different formats that can be accepted by heterogeneous devices, storage and processing for redirecting messages to preferred device that users designate, and heterogeneous device name translation and mapping which may be based on naming schemes other than IP. Like SIP, service hand-off and Universal Inbox are targeted toward telecommunication services.

Mobile People Project [5] utilizes Personal Proxy to route communication to a mobile user, independently of the user's location and applications she is currently using. Personal Proxy shares a similar concept of personal data storage as our BSR repository server. In addition to personal data storage, Personal Proxy also plays an active role in tracking mobile people's whereabouts, routing application communications, and transforming communication protocol to the preferred devices designated by mobile people. In comparison, BSR service provides a centralized personal data storage specifically for browser session state preservation and migration, rather than a broader range of personal data.

Roma Project [6] provides centralized personal metadata service to locate and synchronize current version of personal files and to ensure the availability of the personal files across different repositories. It is similar to BSR service in the sense that BSR service is synchronizing the browser state across devices, whereas Roma is synchronizing personal data in general. In Roma architecture, the data can be distributed across multiple devices and repositories. We believe, for the specific purpose of synchronizing web session states, it is more appropriate to keep them in a centralized proxy server for ease of security and data management.

Aglet [8] and Roam system [9] are mobile agent systems that enable an application to migrate at runtime from one device to another device. This runtime migration involves taking a snapshot of application running state on a source device, transferring and restoring the snapshot on a target device. This snapshot approach is also used in our BSR service. However, BSR service differs from these agent-based systems in that it is focused on web applications, and more importantly, it introduces the concept of a personal repository where a user to store multiple snapshots and retrieve them anytime on any device.

3 Design

The design of the BSR service is based on decoupling the association between web sessions (browser state) and a device, in favor of a new association between web sessions (browser state) and its user as shown in Fig. 1. The benefits of this new association are that (1) it allows a user to switch devices in the middle of an active web session without losing it and having to restart it on a new device, and (2) it allows a user to keep track of multiple active web sessions and freely save and continue any active web sessions at any time from any device.

The architecture of BSR service is shown in Fig. 2. It contains 2 modules: *BSR plug-in* and *BSR repository server*. The BSR plug-in is like any browser plug-in that can be downloaded and installed from the Internet. BSR plug-in exposes user interface (UI) as a bar in a browser for users to perform two basic operations:

- Take a snapshot of the current browser state and store the browser state snapshot on BSR repository server in a secure manner.
- Retrieve a saved snapshot from BSR repository server to a browser in a secure manner.

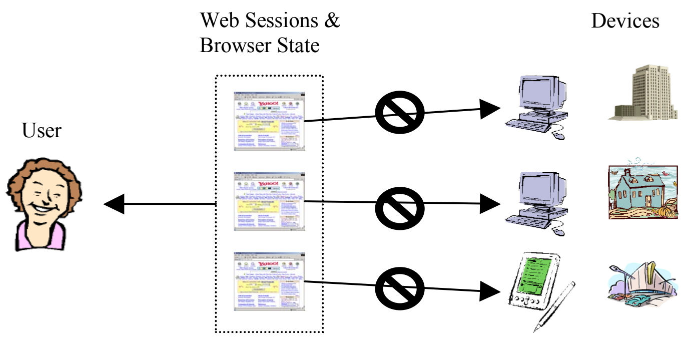


Fig. 1. Design Principle of BSR Service

BSR repository server provides an always-connected storage for authenticated users to store browser state snapshots. In addition, BSR repository server is also a web host that serves the web document displayed in the BSR plug-in bar shown in Fig. 4. By using two basic operations provided by BSR service, Jane can easily migrate an active web session between different devices as shown in Fig. 2. Prior to leaving the source device (device A), Jane takes a snapshot of the browser state and stores the snapshot on BSR repository server. After the target device (device B) becomes accessible to her, Jane retrieves the browser state snapshot from BSR repository server, and restores it on the browser on the target device (device B).

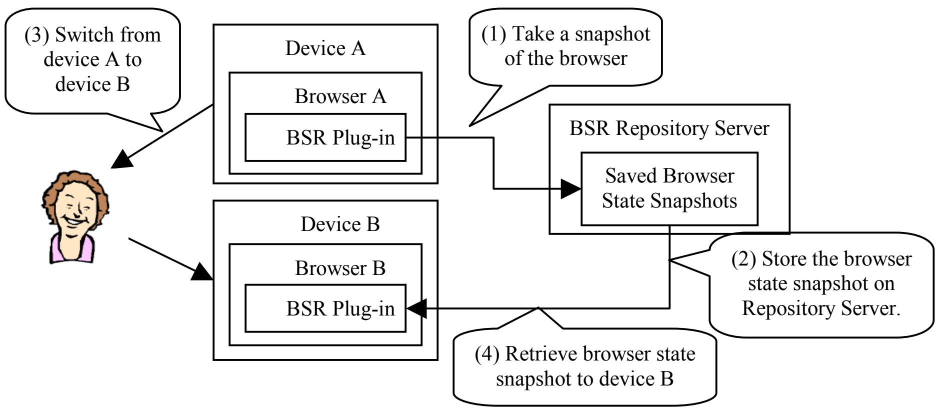

(3) Switch from device A to device B
Device A
Browser A
BSR Plug-in
(1) Take a snapshot of the browser
BSR Repository Server
Saved Browser State Snapshots
Device B
Browser B
BSR Plug-in
(2) Store the browser state snapshot on Repository Server.
(4) Retrieve browser state snapshot to device B

Fig. 2. Web Session Preservation Architecture

For the remainder of this section, we provide a detailed step-by-step description of BSR service corresponding to the example in Fig. 2. Jane first needs to activate the BSR plug-in when she starts her web browser on device A as shown in the first step of Fig. 3. The steps needed to activate a browser plug-in differ from one web browser to another web browser. In Microsoft Internet Explorer (referred to as IE for remainder of this paper), BSR plug-in can be found and activated by this sequence of browser menus: View -> Explorer Bar -> BSR. By activating BSR plug-in, a horizontal explorer bar appears at the bottom of a browser window as shown in Fig. 4. The horizontal explorer bar displays a HTML page downloaded from her designated BSR repository server. We assume that Jane has already downloaded and installed BSR plug-in in her browser, and she has created an account with BSR repository server. From the horizontal explorer bar (referred to as BSR plug-in bar for remainder of this paper), Jane can sign on with BSR repository server by entering her user ID and password and then click on sign-on button. Then BSR plug-in downloads a list of saved browser state snapshots from the BSR repository server. Jane has an option to select either a commercial BSR service provider, or she can setup her own BSR repository.

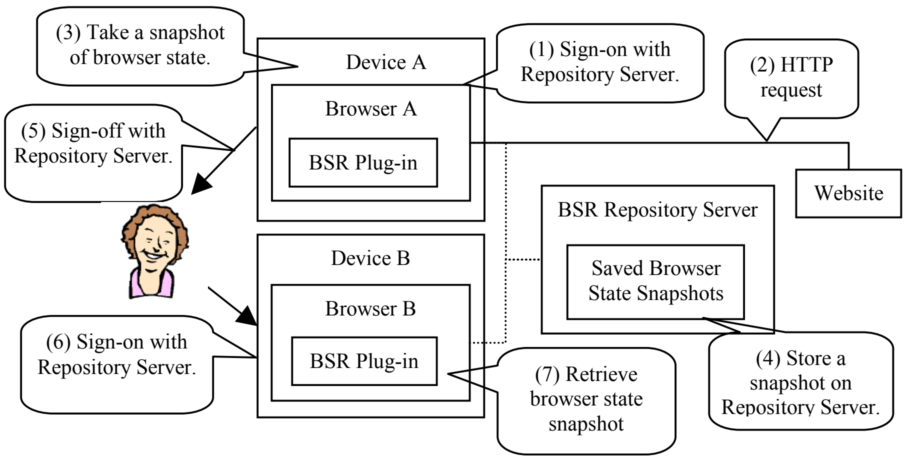

(3) Take a snapshot of browser state.
Device A
(1) Sign-on with Repository Server.
(2) HTTP request
Browser A
(5) Sign-off with Repository Server.
BSR Plug-in
Website
BSR Repository Server
Device B
Saved Browser State Snapshots
Browser B
(6) Sign-on with Repository Server.
BSR Plug-in
(7) Retrieve browser state snapshot
(4) Store a snapshot on Repository Server.

Fig. 3. A Step-by-Step Description of Browser Session Migration from Device A to Device B

Jane can now browse a website on browser/device A as shown in steps (2) of Fig. 3. After some time, Jane wants to leave device A and switches to device B. As shown in step (3) of Fig. 3, Jane first clicks on the snapshot button in BSR plug-in bar. Then BSR plug-in takes a snapshot of the browser session. The snapshot includes the document objects (DOM) of the current web page on the browser, scripting objects (JavaScript or VBScript), URL history, and cookies corresponding to the domain of website(s) serving the current web page. The snapshot is stored securely on BSR repository server in step (4) of Fig. 3 via a SSL connection. Jane can assign a unique *session name* to the snapshot as an index in the list of stored snapshots. If Jane does not give a session name, a default one is generated which is the website hostname. Jane also has an option to protect this snapshot with a *session password*. The session password offers additional protection in the case that Jane carelessly leaves her device open to other people without signing off with BSR service.

Before Jane closes browser on device A, she clicks on sign-off button in the browser plug-in bar. It disconnects the browser from BSR service as shown in step (5) of Fig. 3. Jane is now ready to use device B. As shown in step (6) of Fig. 3, Jane first signs on with BSR service on device B. The BSR plug-in downloads a list of saved snapshots from BSR repository server. Jane selects the session that she was last working on device A. Then the BSR plug-in downloads the snapshot and restores it on the browser as shown in step (7) of Fig. 3. Now Jane has successfully migrate her session from device A to device B. In the previous example, we make an assumption that the web session in the snapshot has not been time-out by the web application server when Jane restores it. However, this may not be true for some web application servers that set short time-out policies when they detect user inactivity. We propose the following solutions to help keep alive any saved web session:

- The BSR service can periodically ping the web application server (e.g., by refreshing the web page stored the browser state snapshot) in order to keep alive a saved web session. It requires no modification to the web application servers, but it has undesirable effect of generating additional network and server loads.
- The BSR service can probe the websites for the session time-out value, and inform its user to retrieve the browser state snapshot before time-out occurs. In addition, the websites can be modified to allow the BSR service to set time-out policies on saved web sessions.

4 Implementation

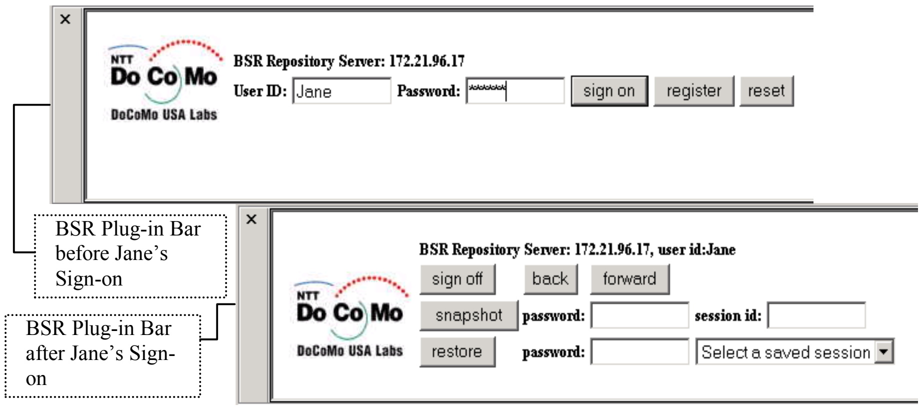
We describe how to implement for the BSR plug-in, in IE version 5.0 and later, to capture and restore a browser state snapshot. The data structure for the browser state snapshot includes the following components – document object state, cookies, script state, and browser history.

4.1 Document Object State

W3C defines a standard specification for document objects called DOM (document object model) [2]. DOM describes a logical structure of web documents and standard interfaces for access and manipulation of web documents by scripting. When a browser parses a HTML page, it creates a DOM structure to represent the structure of a HTML page. Each node in the DOM structure represents a DOM element, which in turn may represent a particular HTML tag or a HTML element defined in DOM specification, such as <HEAD>, <BODY>, <SCRIPT>, <FRAME> and etc. Each DOM element has a certain set of properties that describe its presentation and behavior in the browser. In a dynamic HTML page, properties of DOM elements and DOM structure are its runtime state that can be changed by scripting when a user interacts with the page. As a result, there is a need to capture this DOM runtime state of a web page in a browser state snapshot.

We have designed two possible methods for capturing and restoring document objects called *URL reloading method* and *content reloading method*. The URL reloading method captures the DOM runtime state of a web page in a browser and the URL(s) to this web page. The DOM runtime state is extracted by traversing each node in the DOM structure of a web page and serializing its values and properties. When a snapshot is restored, the web page is first loaded from the original website specified in the saved URL(s). After the web page is completely downloaded in the browser, the retrieved DOM runtime state is de-serialized and applied to the web page. For example, when Jane enters her name "Jane" on a text field of a HTML page from the online drapery store (e.g., <INPUT NAME= "name" TYPE="text">), the browser assigns the value property of the DOM text field element to "Jane". When Jane takes a snapshot, BSR plug-in serializes the value property of the text field DOM element and stored it on the BSR proxy. In a restoration, BSR plug-in reloads the HTML page from the online drapery store, which initializes the text field to be empty. Then it de-serializes the DOM runtime state, applies it to DOM text element, and restores the text field back to "Jane".

The content reloading method captures the content of the web page as displayed on the browser. In a later restoration, it retrieves the content from the BSR repository server to the browser rather than reloading it from the original website. There is no need to serialize the DOM runtime state because the saved content contains modified runtime state. Consider the same example which Jane enters her name "Jane" on a text field of a HTML page from the online drapery store. When Jane takes a snapshot, the browser plug-in saves the modified HTML page including her input name, e.g., <INPUT NAME="name" TYPE="text" VALUE="Jane">.


BSR Repository Server: 172.21.96.17
User ID: Jane Password: ******

BSR Repository Server: 172.21.96.17, user id:Jane

Fig. 4. BSR Plug-in Bar before Jane's Sign-on and After Jane's Sign-on

The main difference between these two methods is that when a snapshot is restored, URL reloading method generates request(s) to the original website, whereas the content reloading method does not generate any request(s) to the original website. However, the URL reloading method works better for web pages that are updated periodically. When Jane restores a snapshot, URL reloading method can retrieve the most updated content from the website; whereas the content reloading method may display content that have expired already.

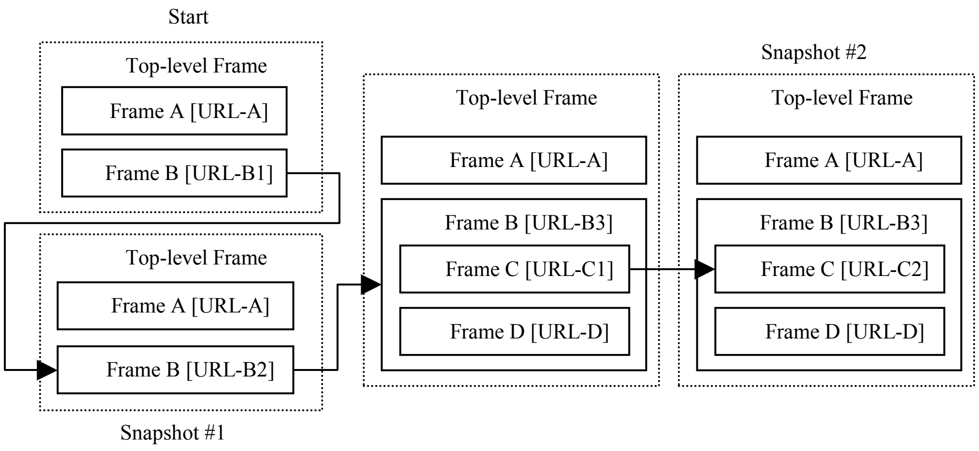


Fig. 5. Keeping Track of URLs to Frames in a Web Page

4.2 Tracking Frames

A web page can contain multiple frames and sub-frames. Content in each frame or sub-frame can be changed at runtime by scripting to load from a different URL/website. Since frames are also document objects, tracking runtime state of frame and sub-frame is a necessary part of capturing the overall DOM runtime state of a web page. Consider the example in Fig. 5. The web page contains two frames – an index frame A and a content frame B. When the web page is first downloaded to a browser, URL-B1 is loaded in frame B. When Jane clicks on a link in frame A, frame B is reloaded with URL-B2. At this time, Jane takes snapshot #1 of the browser state. In URL reloading method, the browser plug-in creates a *frame-URL-mapping table* for snapshot #1. In this table, each entry contains the name of a frame and the *modified URL* to the frame (Frame B, URL-B2). When the user restores snapshot #1, browser plug-in traps all HTTP requests and checks if frames that HTTP requests are originated from match entries in frame-URL mapping table. In case of a match, the

URL in the HTTP request (URL-B1 for frame B) is replaced with the modified URL (URL-B2 for frame B) recorded in the frame-URL mapping entry. A more complex example is shown in snapshot #2 of Fig. 5, where frame B is loaded with yet another frameset (URL-B3) that contains two sub-frames C and D. Frame C is first loaded with URL-C1, and reloaded with URL-C2. In snapshot #2, the frame-URL-mapping table contains two entries where URLs have been modified {(Frame B, URL-B3), (Frame C, URL-C2)}. When a user restores snapshot #2, HTTP requests for URL-B1 and URL-C2 are replaced with URL-B3 and URL-C2.

4.3 Cookie State

Cookies are information stored on client browsers for specific domains of web documents. Cookies can be used for many different purposes, such as keeping track of a web session with a web application server, storing intermediate session state, identifying the user between sessions, and etc. When a snapshot is taken on a source browser (device) and restored on a different target browser (device), the snapshot may not work on the target browser if cookies are not identical between two browsers. For example, a web application server may set a cookie containing the session ID on the client browser. If the session ID cookie were not set on the target browser, the web application server would not be able to identify the session from HTTP requests sent by the target browser.

A simple but inefficient method is to save all cookies on the source browser in a browser snapshot, regardless of domains of cookies, and transfer them to the BSR repository server. When the browser snapshot is restored, all cookies are transferred to the target browser. This is inefficient because only the cookies that match domains of the websites in the snapshot are used on the target browser. A more efficient method is to identify domains of websites in the snapshot, and transfer only the yet expired cookies matching those domains.

4.4 Script State

Popular client-side script languages embedded or referenced in HTML pages are JavaScript and VBScript. Client-side scripts run on a client browser, and they can be used to manipulate the appearance and content of a dynamic HTML page without having to go to a web application server. Like many programming languages, client-side scripts contain script methods and script variables. The scope of a script variable can be either local (declared within a script method) or global (declared outside of any script methods). In a browser state snapshot, BSR plug-in does not need to save local script variables because they are reinitialized each time a script method is invoked. In addition, BSR plug-in makes sure that it does not take a snapshot in the middle of script method execution where the execution stack is not empty. It waits to take a snapshot until the script engine of the browser has completed executing script methods. However, BSR plug-in does need to save global script variables because their values remain valid until a browser navigates to other web pages.

BSR plug-in does not need to save any script methods, because script methods do not change and they are reloaded into a browser when a snapshot is restored.

4.5 History State

Browser history state can be an essential part of a browser state snapshot, which is used to support the back and forward buttons found on almost all browsers. BSR plug-in maintains its own *history state* for a browser state snapshot, and it is implemented independently from the browser history mechanism.

According to the HTTP 1.1 protocol specification [7], browser history should show exactly what a user saw at the time that the web page was displayed in the browser. It differs from browser cache mechanism in that if a web page retrieved from history has expired, browser should still display the expired web page as is in the browser. On the other hand, if the same expired web page is retrieved from the browser cache mechanism, browser should re-fetch the updated copy from the original website. This means that BSR plug-in needs to save the actual content of historical web pages in a snapshot. Saving URLs of historical web pages is insufficient. As a result, only the content reloading method described in section 4.1 can be applied to saving history state but not the URL reloading method.

BSR plug-in starts recording browser history when it is activated in a browser, so history of a snapshot can only go back to the web page where the BSR plug-in is activated. However, the amount of data for history state can be huge at the end of a long browsing period. It can become too big to be transferred to and from a BSR repository server in a snapshot or a restoration. To address this issue, BSR plug-in has an adjustable limit on the size of the most recent history that it keeps track of. If the history state grows beyond its limit, older web pages are thrown out. In addition, for mobile devices where bandwidth is a premium, history state is downloaded only as needed from repository server to a mobile device when the user clicks on the back button. A user can also disable history state tracking completely on BSR plug-in

4.6 Performance

The performance of the a snapshot or restore operation depends on several factors – the type of network connectivity, the size of snapshot which is proportional to complexity of web pages and the size of the history, and the load on the BSR server. We have conducted preliminary testing of snapshot and restore operations on various popular web sites (such as amazon.com and yahoo.com). The size of the snapshot is approximately equal to the size of the downloaded HTML page(s). If history tracking is enabled, the size of the snapshot is approximately the sum of all HTML pages recorded in history. We have found that on average, a snapshot or a restore operation, with history tracking disabled, takes less than 5 seconds on a desktop PC with a fast WLAN/LAN connection, with the majority of the time in the serialization and de-serialization of the DOM tree.

5 Conclusion and Future Work

In this paper, we describe BSR service that brings browser session mobility to users. BSR enables a user to switch devices in the middle of an active browser session. In addition, it enables a user to preserve multiple active web sessions for restoration at a later time on any device with a built-in browser support. We believe that as browser-based applications gain popularity, browser session mobility provided by BSR service can help browser-based applications to better integrate with people's mobile work and life styles.

We expect such browser session mobility to occur not only among devices and/or browsers with similar capabilities, but also across heterogeneous device platforms with different browsers/micro-browsers (HTML, cHMTL, WML, and etc.) and hardware capabilities (screen sizes, input methods, and etc.). This introduces additional challenges. For examples, the same web page can require different presentations (HTML, cHTML, or WML) on different browsers, so snapshot on one device platform may require transformation before it can be restored on another device platform. Browsers and micro-browsers also differ on cookie management, the type of scripting language, and history management. BSR service for multiplatform applications is our future work.

References

1. Henning Schulzrinne and Elin Wedlund, "Application-Layer Mobility Using SIP", *Mobile Computing and Communications Review*, Volume 4, Number 3, pp. 47--57, July 2000.
2. W3C, "Document Object Model (DOM) Level 1 Specification (Second Edition)", W3C Working Draft, September 2000. http://www.w3.org/TR/2000/WD-DOM-Level-1-20000929
3. Anthony Joseph, B. R. Badrinath, and Randy Katz, "A Case for Services over Cascaded Networks", *First ACM/IEEE International Conference on Wireless and Mobile Multimedia (WoWMoM'98)*, October 30, 1998.
4. Bhaskaran Raman, Randy H. Katz, and Anthony D. Joseph, "Universal Inbox: Providing Extensible Personal Mobility and Service Mobility in an Integrated Communication Network", *Workshop on Mobile Computing Systems and Applications (WMSCA'00)*, December 2000.
5. P. Maniatis, M. Roussopoulos, E. Swierk, K. Lai, G. Appenzeller, X. Zhao, and Mary Baker, "The Mobile People Architecture". *ACM Mobile Computing and Communications Review*, Volume 3, Number 3, July 1999.
6. Edward Swierk, Emre Kicman, Nathan Williams, Takashi Fukushima, Hideki Yoshida, Vince Laviano, Mary Baker, "The Roma Personal Metadata Service", *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications*, December 2000.
7. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1", RFC 2616, June 1999.
8. D. B. Lange, M. Oshima, "Mobile Agents with Java: The Aglet API", *World Wide Web Journal*, 1998.
9. Hao-hua Chu, Henry Song, Candy Wong, and Shoji Kurakake, "Seamless Applications over Roam System", *UbiTools'01 (Part of UbiComp'01)*, September 2001, http://choices.cs.uiuc.edu/UbiTools01/.