4-11-2011

Implementing Time Travel for the Web

Robert Sanderson
Research Library of Los Alamos National Laboratory

Harihar Shankar
Research Library of Los Alamos National Laboratory

Scott Ainsworth
Old Dominion University

Frank McCown Ph.D.
Harding University, fmccown@harding.edu

Sam Adams
University of Cambridge

code{4}lib JOURNAL

Issue 13, 2011-04-11 ISSN 1940-5758

Implementing Time Travel for the Web

This article discusses the challenges and solutions discovered for implementing the Memento protocol in a variety of browser environments. It describes the design and deployment of the client technologies which have been developed: a web application that functioned as a browser, an add-on for FireFox called MementoFox, a plugin for Internet Explorer and an Android-based client application. The design and technical solutions identified during the development will be of interest to those considering implementation of a Memento based platform, especially on the client side, however the interactions are also important for building conformant server-side systems.

By Robert Sanderson, Harihar Shankar, Scott Ainsworth, Frank McCown, and Sam Adams

Introduction

Developers of digital libraries and archives built for digital preservation are very aware of the time dimension when it comes to their constituent resources; document models in digital libraries are well stocked with different versioning paradigms, and maintaining old versions of documents is the foundational aspect of digital archives. Multiple timestamps, different creators and editors, and different publication statuses are just some of the aspects that need to be dealt with by any such repository of knowledge. It is well understood that access to prior versions of the documents is important for the scholarly record and authenticity of research.

Recent trends in digital libraries are towards integration with the architecture of the World Wide Web, either through Web 2.0 functional behavior or with semantics via Linked Data and RDF. One feature of both is the use of HTTP content negotiation [1] with “Cool URIs” [2] that provide access to the same resource in different formats, or provide embedded functionality that might previously have required multiple pages and search forms.

These URIs stay constant even though the representation delivered in the response changes. This is what makes the web possible in a distributed environment; if each different state had a different identifier, the links between pages that make up the Web would break every time a new version was introduced. The Memento project [3] proposes to combine the understanding of the importance of time and versions possessed by the digital library and preservation community with current thinking in the Web domain. It should be possible for a browser to automatically navigate from the original resource to archived versions of its previous states using negotiation and other protocol level operations, rather than having to know in which archive (such as the Internet Archives [4] or the UK National Archives [5]) to search for them by hand.

This article first describes the protocol level details and then discusses the challenges of a client side implementation for the Firefox web browser. Some early work on an equivalent Internet Explorer plugin is presented, along with the implementation of an Android application called Memento Browser. It is hoped that this research will provide insight into the practical issues of Memento and motivation for further implementations that enable seamless access to the historical web.

Memento

There are three main issues, hinted at above, with existing access to archived resources on the web:

1. Instead of being accessible directly or indirectly from their original identifier (their URI), the archived resources are available from new locations with no method to discover the most appropriate version, given their original URI. Even if the user does know of one or more archives in which to search, the most appropriate version for their need may be in an archive that is unknown to them.
2. Each archive provides a different, manual search and selection interface that the user must navigate to discover the desired resource. However, given only the Original Resource’s URI and the desired time, the browser, if it understood the navigation process, could use this information to take the user straight to the appropriate resource without manual interaction.
3. Once the user has found an archived version of the resource, future navigation is inconsistent. Either: (a) the navigation is locked into a single archive due to URLs being rewritten, potentially to resources which do not exist or are not the most appropriate for the user’s requested timestamp; or (b) if the URLs are not rewritten, the user will instantly be taken out of the

past to the current state of a resource rather than an appropriate archived copy, as the browser does not know of the user's desire to browse the past versions of the resources.

Memento approaches these issues with a protocol level solution that can be implemented by clients, servers and intermediate services. The currently evolving specification is available at the MementoWeb site [6] and the details are summarized below.

The first step in making previous states of resources discoverable at a protocol level is to make the archived resources indirectly available from the Original Resource's URI, *R*. Whenever the Original Resource is dereferenced, the web server can add an HTTP `Link` header [7] to the response which points to a new type of resource called a TimeGate, *G*. This response can be the same for any request, as the header makes no other impact on the transaction for clients unaware of Memento. The request can be made as a HEAD, as this link is the only information of interest in the response.

The TimeGate provides the solution for the second issue, as it understands requests with a timestamp included in a request header, and can redirect the client to the most appropriate archived version that it is aware of without manual intervention. The client adds an `Accept-Datetime` header containing the timestamp of interest in RFC 1123 format [8], the mandated timestamp format for HTTP, to the request. The server responds with a 302 status and a `Location` header pointing at the appropriate archived resource, which we call a Memento and is depicted as *M* in Figure 1. The TimeGate also adds in links to the Original Resource, along with the first and last archived versions it knows of, and optionally to other Mementos it thinks may be useful to the client, such as the next and previous in the sequence.

The client then follows this redirection and ends up at the appropriate archived resource for the user's request. The archival server includes in a `Memento-Datetime` header, the timestamp at which it knows the representation was served from the Original Resource's URI. The archival server should also provide links to both *R* and *G*.

This transactional process is applied to all requests made by the client and as such provides a solution to the third issue, as embedded resources, such as images or stylesheets, are resolved in the same time-aware fashion as the encapsulating HTML page without requiring the URIs to be rewritten. In fact, having the URIs rewritten is not necessarily the optimal strategy as the client stays locked within the current archive rather than potentially having access to resources across multiple archives. This series of interactions is depicted in Figure 1 below.

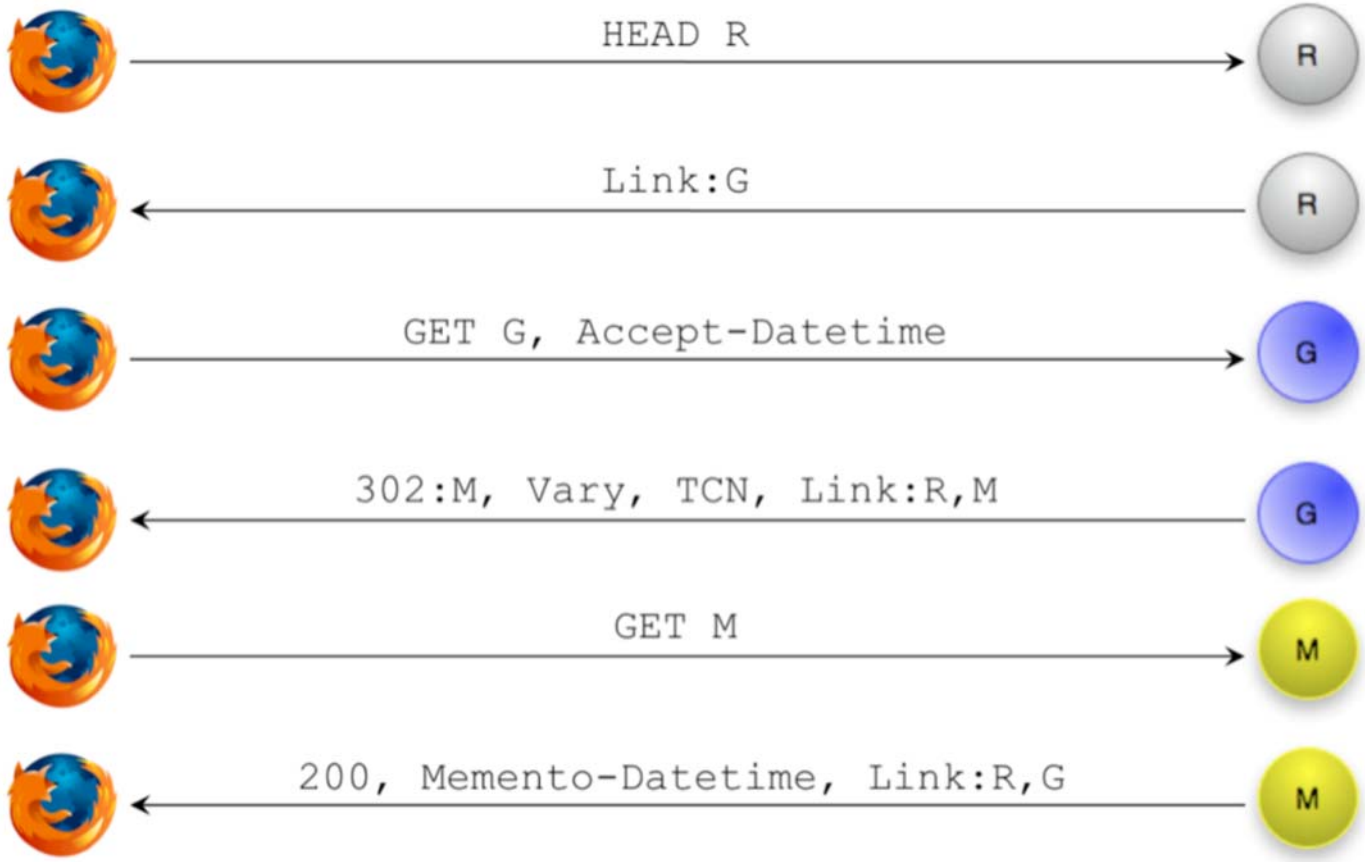

Figure 1: Memento HTTP Transactions
R = Original Resource; G = TimeGate; M = Memento

Memento also defines an API for interacting with archives in order to discover their holdings for a particular Original Resource. There are two complementary implementations of the API, one is aligned with Linked Data and based on OAI ORE [9], and the second uses the same CSV-like format as the `Link` header for ease of processing. The document retrieved is called a TimeMap and at a minimum includes the URIs of the archived versions and the time for which they are appropriate. In the ORE implementation, a TimeBundle is a specialization of an ORE Aggregation that aggregates all of the known archived versions of a given resource. The TimeBundle and TimeMap are referenced from the TimeGate via an entry in the `Link` header in the response, and the client can thus choose which type is easier for it to process. The information from the TimeMap can then be used by third parties to create overlay services, TimeGates with access to the holdings of multiple archives, and to provide more detailed client/server interactions.

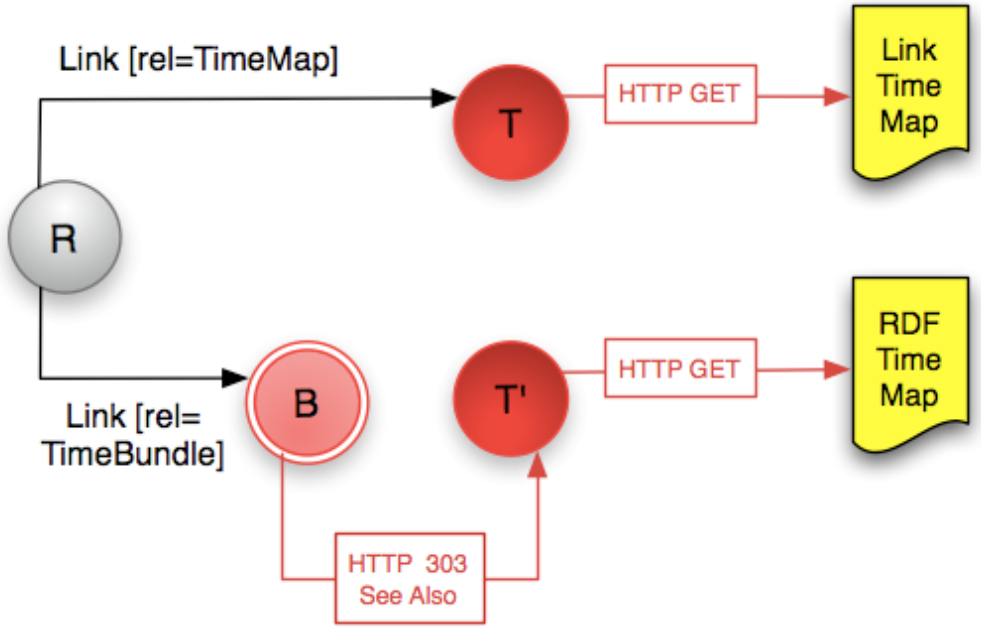

Figure 2: Time Map API
R = Original Resource; B = TimeBundle; T and T' = TimeMap

Server Side Memento Client

In order to demonstrate that the Memento solution described above would work, it was necessary to implement a client capable of performing the request/response cycle of Figure 1, and understanding and acting upon the headers involved in the transactions.

The first attempt at building a client resulted in a web application,implemented and hosted at Los Alamos National Laboratory, which reconstructed the requested page from known web archives. [10] Using this client, the user starts by submitting the URL and the time of the desired version to a simple web form. The client then performs all of the necessary HTTP transactions for Memento, following the `Link` header and redirects in order to obtain and locally cache the archived representation. If the retrieved resource is an HTML document, then the client parses it and steps through each URL, such as images, stylesheets and JavaScript, fetching the appropriate archived representation and caching it recursively to deal with frames. When there are no more URLs to fetch, it returns the requested resource to the browser to be displayed to the user. This architecture is depicted below in Figure 3.

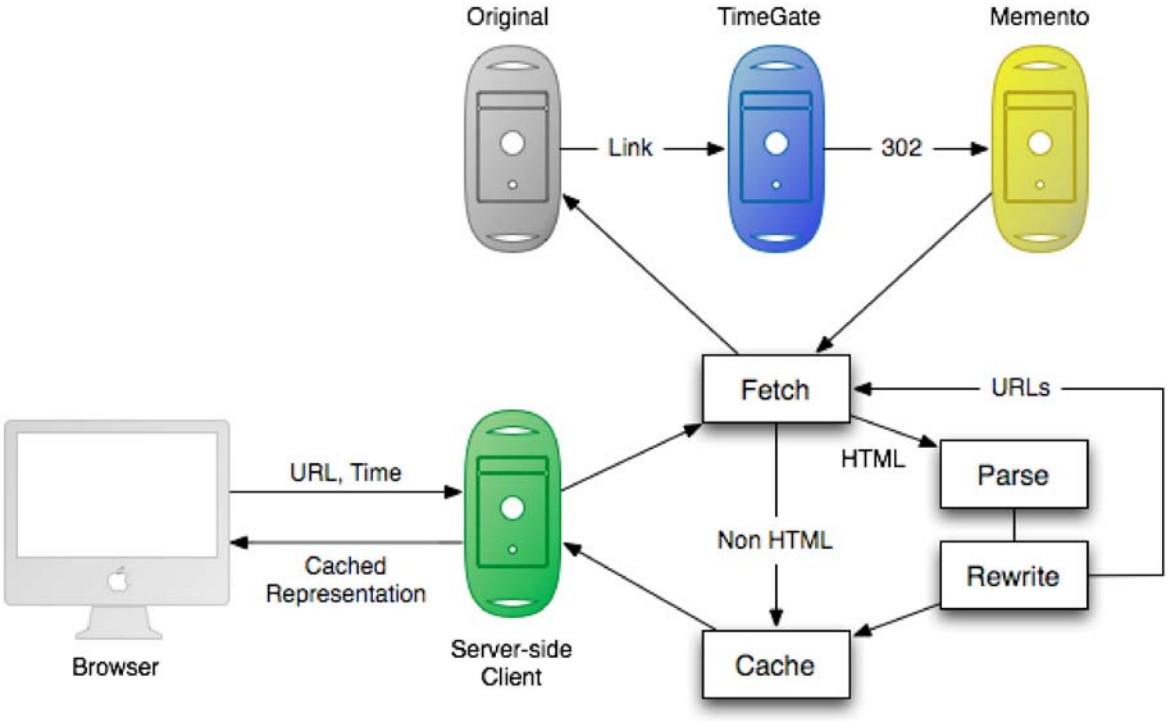

Figure 3: Serverside Client Architecture

While informative and tolerable as an initial demonstration, the web application was slow and unreliable as it was necessary to cache all of the component resources locally before returning the page to the user, thereby making incremental rendering of the page impossible. Not only was the user stuck inside a single, albeit distributed, solution, it exhibited all the problems associated with HTML parsing and URL rewriting, from bad HTML to links being generated dynamically with JavaScript. On the other hand, it did make it easy to track usage and get a feel for how, or even if, people would use a truly integrated client. The more than 10,000 sessions logged in the two month lifetime of this little application gave us great motivation to create MementoFox.

Memento State Machine

Having already run into many unexpected issues in the server-side implementation, we were able to describe a state machine for all of the possibilities that a Memento client would encounter. As depicted in Figure 4, the logic requires access to the response status code and headers at each step to determine the next action.

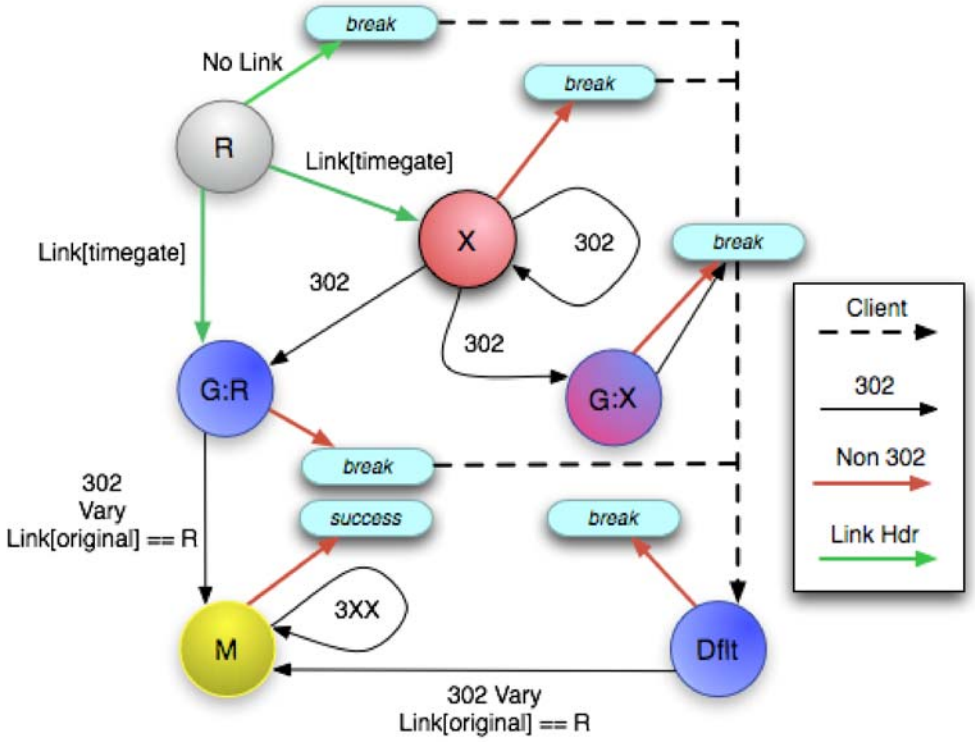

Figure 4: Memento Client State Diagram
R = Original Resource; G:R = Time Gate for R; M = Memento for R; X = Another Resource; G:X = Time Gate for X; Dflt = Default Time Gate

Starting with the Original Resource *R*, there may or may not be a `Link` header with a `rel` attribute of `"timegate"`. It is accepted that in the early days of Memento adoption, the vast majority of web servers will not know of a web archive that contains Mementos of its hosted resources, and hence will not include a link to a TimeGate. If the `Link` header is not present, it is treated as an (expected) error condition and the client should use its default TimeGate *Dflt*. If the `Link` header does exist, the client should follow it to either a real TimeGate for the resource *R*, the node *G:R*, or in the case of a misconfigured server, to another resource *X* which is not a TimeGate for *R*. A TimeGate can be detected by inspecting the `Vary` header of the response for the string “accept-datetime”. The resource that the TimeGate is for is given in the `Link` header with a `rel` attribute of `"original"`.

If the link takes the client to a resource that is not a TimeGate, that resource will either issue a response with a 302 status code or some other status code. In the case of a non-302 response, an error condition is reached and the client should take over again by using its default TimeGate. On the other hand, if the response does have a 302 redirect, there are three possibilities: it can be to another resource that exhibits the same behavior as *X*, to a TimeGate *G:X* that in turn redirects to a Memento of *X* rather than a Memento of *R*, or to a TimeGate *G:R* for the Original Resource *R*. The first two 302 conditions result in the error state; however the admittedly unlikely final case would result in the client continuing its normal behavior.

In the case where the `Link` header from the Original Resource is to a real TimeGate for *R*, *G:R*, then it can either issue a non-302 response, resulting in an error condition, or issue a 302 with the `Vary` header and a `Link` header with `rel` attribute of “`original`” pointing back to *R*. This `Link` header is necessary to determine the difference between *G:R* and *G:X* and the `Vary` header is necessary in order to determine that the redirection is due to time-based content negotiation.

The default TimeGate may also generate an error condition, at which point the client must either continue to try default TimeGates until one provides a correct response or stop its search. Finally, either via *Dflt* or *G:R*, the client will arrive at a Memento, *M*. This may in turn redirect with a 3XX status to another resource, such as for a Memento that has been moved to another archive. Otherwise, the client has successfully reached the end state.

MementoFox

Initial Approach

Mozilla's Firefox [11] was chosen as the first browser platform as building extensions for it is relatively straightforward. They can be implemented in JavaScript and distributed without requiring Firefox be recompiled or the extension to be compiled for different operating systems. There are tens of thousands of registered add-ons which can be used as exemplars [12], and the documentation for the inner workings of the browser is clear, up to date and easily available [13]. The user interface for the extension is described in an XML schema called XUL, which Firefox interprets at run time. Any other required files, such as images, are placed in an extension-specific directory. There is easy access to important internal functions, such as preferences, URL and HTTP protocol functionality via a JavaScript/C++ reflection layer called XPCOM. Once finished, the add-on is packaged with all of its resources in a zip file called an xpi. When Firefox sees one of these files, it knows to install it rather than attempt to render the resource.

Our initial approach, implemented at Old Dominion University, created a sidebar for user interaction and relied on the obvious Firefox events that monitor and change HTTP requests. The sidebar allowed the user to enable and disable Memento, specify the target date and time, and view the results of Memento operations. Figure 5 is a screenshot of the initial approach. To implement the Memento protocol, two events were used. The `http-on-modify-request` event was used to add the `Accept-Datetime` header to the request. The sidebar and extension's internal state were updated on the `http-on-examine-response` event. This approach was sufficient for use with Memento-aware HTTP servers but could not detect and respond to responses from services that were not yet Memento-compliant.

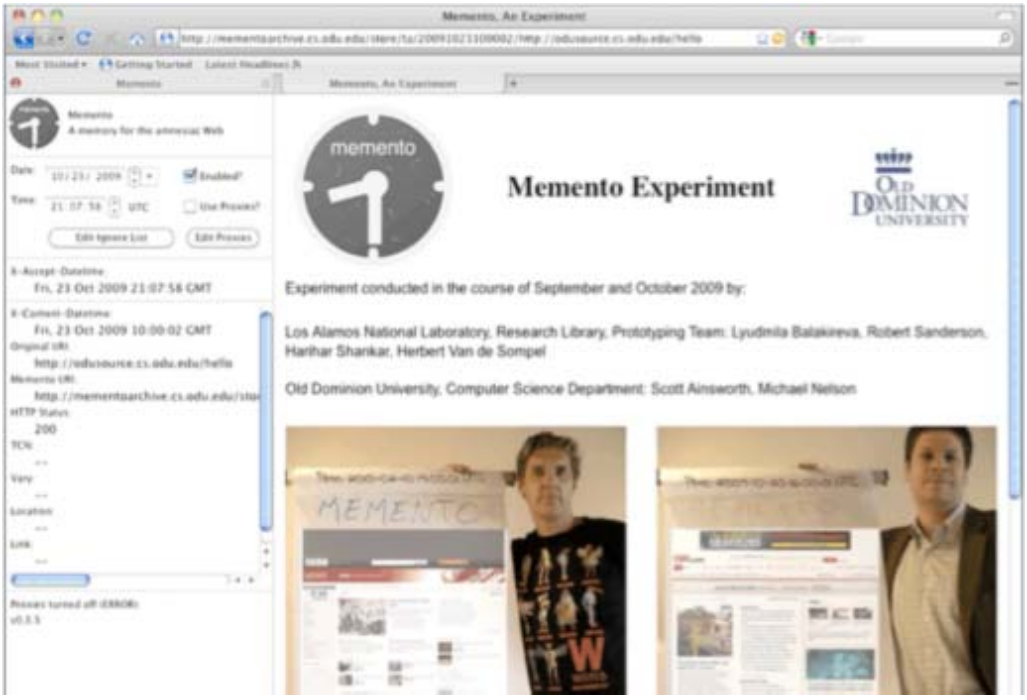

Figure 5: Initial Approach Screenshot

Monitoring the `EndLoadDocument` event allowed non-Memento responses to be detected and acted upon. When a non-Memento response was detected, the extension changed `document.location` to a TimeGate at Los Alamos National Laboratory that searches a number of archives, thus satisfying the user's request when the original server could not. Unfortunately, `EndLoadDocument` is triggered after a page's embedded and internal resources are fully loaded, resulting in a confusing user experience in which the embedded resources appeared in their current state and were then replaced with their archived versions. Aside from discovering the limits of Firefox's HTTP monitoring events, our initial approach contributed a method to apply Memento independently for each Firefox content window. This resulted in a working client, but one that provided a poor user experience. While this implementation was faster and closer to the desired result than the server side client, a solution to the display issue proved to be elusive.

FireBack

Elusive, at least, until JISC sponsored a competition to develop Memento-related software at the Developer Happiness Days event in London at the end of February 2010 [14]. The winning entry, called FireBack, was a Firefox add-on that demonstrated the way around the obstacle. Instead of capturing the response, FireBack used an interface called `nsIContentPolicy` in order to prevent Firefox from making the request in the first place. This interface is normally used to block content such as unwanted advertising or resources unsafe for viewing by children, and was discovered by examination of the Adblock Plus add-on [15], which requires such behavior.

The `nsIContentPolicy` interface is designed to allow an add-on to observe content as it is loaded into the browser and enable the add-on to block or permit access to particular URIs. The interface provides two methods: `ShouldLoad()`, which is called before a resource is loaded and allows the add-on to decide whether to start loading the resource, and `ShouldProcess()` which allows the add-on to inspect a resource's content and decide whether to render it. Experimentation found that as well as being able to reject access to a requested resource from within the `ShouldLoad()` method it was possible to direct the browser to load an alternative resource.

This approach worked well, albeit with one usability issue: loading a page resulted in the UI locking up, with the window completely frozen to user input for ten or more seconds at a time. Investigations found that Firefox is a largely single-threaded application, and while the initial HEAD request to the Original Resource is typically very fast, the GET request to the TimeGate can be fairly slow. The synchronous nature of the GET request was causing Firefox's main thread to block until a response was received from the TimeGate.

It was thought that the delay could be minimized if only the HEAD request were performed during the invocation of the `ShouldLoad()` method, the browser then instructed to load the TimeGate URI, and the browser's internal redirect handling mechanism relied on to handle the redirection from the TimeGate to the Memento. Unfortunately, while it is possible to direct the browser to load an alternative URI from within the `ShouldLoad()` method, it is not possible to access the HTTP headers of that request and add the `Accept-Datetime` header.

The solution to add the headers was to employ the event observation technique of the initial approach. The header was added by processing the `http-on-modify-request` event, and cleanup performed on the corresponding `http-on-examine-response`. This considerably reduced the scope for the browser's main thread to block, leading to a much more responsive UI.

Current Version

The FireBack networking solution and the multiple tab/window handling abilities of the initial add-on were combined with improved internal logic and user interface to form the basis of the current version of MementoFox. The majority of the MementoFox logic takes place in `ShouldLoad()`, which occurs immediately before the URL is retrieved. If the resource should be loaded, the function returns ACCEPT, otherwise it returns one of four different REJECT flags. Determining whether to accept or reject and rewrite a request is the fundamental issue for Memento client development.

The function has a series of cases that should always be accepted, for example, if the protocol is not HTTP, if the resource is a Memento, or if it has already been rewritten by the add-on. It then follows the logic described above in order to find the most appropriate archived version. As discovered in the implementation of FireBack, it is possible to modify the URL being processed in place, and this allows us to internally redirect the browser to the appropriate TimeGate for the top HTML page or to the Memento for embedded resources such as images. Firefox processes embedded resources in the context of the encapsulating HTML element (or the `nsIDOMNode` to be exact) that refers to them, rather than at the browser level in order to ensure the correct rendering. This means that the DOM node must be modified internally to point to the Memento, in a kind of client-side URL rewriting operation, but one which occurs before the original resource is retrieved.

If the Memento comes from an archive which does not yet support Memento natively, detectable via the lack of `Memento-Datetime` header, then in a pre-processing step, the system will first attempt to *unrewrite* any URIs that have been rewritten to point to versions within the archive. Current web archives will rewrite, for example, a link to Google with a link to an archived version of the Google homepage in order to provide an appropriate user experience. This is a temporary workaround in order to use non-compliant archives as they are currently, but is not possible in all situations as it relies on implementation-specific heuristics.

The add-on must also implement an event observation method in order to capture the headers of the HTTP responses. This `http-on-examine-response` event is what allows us to find the `Link` and `Vary` headers. This information lets us warn the user if the headers look suspicious and to provide additional navigation options, such as buttons to go to the first, last, previous and next Mementos. Other than adding the `Accept-Datetime` header on the outgoing requests, the observer is also used to update preferences in real time.

After releasing the first public version, we received a lot of very positive and constructive feedback about the user interface. In particular, the time slider was well received and the menu bar rather than sidebar style of the first attempt was appreciated. However, users found our initial terminology confusing, and wanted even less of an interface footprint: VCR-style buttons for first, last, previous and next replaced a long drop down list, and the menu bar can be automatically hidden when the add-on is inactive. Additional functionality was added, such as control for the behavior when an appropriate Memento cannot be discovered: a preference determines if the current version or an error message is displayed.

It was necessary to then rewrite a large proportion of the codebase in order to bring it up to the standards required by Mozilla for the add-on to be endorsed as safe. Variables in add-ons are not inherently encapsulated within the code, so any global variable can unexpectedly collide with another add-on's variable of the same name. This was corrected by hiding all of the code inside a uniquely named object. All of the user interface strings needed to be exported to an external XML file in order to allow for localization, and

further interface improvements were suggested. As of version 0.8.10 (September 2010), MementoFox has been endorsed by Mozilla as an add-on that is safe for widespread use [16]. The codebase has subsequently been published via the Google Code repository [17] to allow for distributed development.

The screenshot in Figure 6 is the result of dragging the time slider to October 2007 when the browser is displaying the current http://www.code4lib.org/ web page. The MementoFox add-on takes over and discovers an appropriate version in the Internet Archive, via a multi-archive TimeGate chosen in the add-on's user preferences. It retrieves the embedded images and other resources to generate an accurate representation of the historical version, without any manual interaction with web archives.

Figure 6: Screenshot of MementoFox

Internet Explorer

An Internet Explorer plugin, known as a Browser Helper Object (BHO), is a Dynamic Linked Library (DLL) module. The BHO is loaded each time Internet Explorer starts up, runs in the same memory context as the browser and can perform any action on the available windows and modules. The BHO components communicate with IE by implementing a binary interface that enables inter-process communication, called the Component Object Model (COM) interface.

The proof-of-concept Memento plugin for IE is written in C# and uses the .Net Technology. Creating the BHO turned out to be both difficult and time-consuming, particularly as documentation specific to the development task is sparse [18], and there are very few useful examples available.

The Memento BHO implements the `IObjectWithSite` COM interface to communicate with IE and attach to its typical events. In particular, the Memento BHO invokes the `OnBeforeNavigate2` event, which fires before any navigation occurs in the web browser. The BHO observes this event and then follows the logic described earlier to retrieve the Memento. The logic for the Memento BHO varies from MementoFox only when displaying the embedded resources of a Memento. Before IE navigates to the Memento, the BHO intercepts, fetches its DOM, parses it, retrieves the Memento for the embedded resources and renders it in the browser window.

A screenshot of the BHO and its preliminary UI is shown in Figure 7. The BHO currently has a very simple UI with a button to toggle Memento Mode On/Off, a Calendar to choose the date-time and a “Navigate” button to start the Memento navigation. To create the toolbar, the `BandObjectLib` Library is used [19]. The Calendar tool could not be directly incorporated into the toolbar. Instead, the Calendar from Windows Forms had to be wrapped by extending `ToolStripControlHost` in order for it to be used in the IE toolbar. This meant that the calendar tool could not invoke the IE events directly and hence a separate “Navigate” button was required to trigger the navigation after a date was picked. There are no buttons yet in the toolbar to navigate to previous and next Mementos.

Installing the Memento BHO and the toolbar is simple as all of the necessary files are packaged for installation and stored as an MSI file. Installing or uninstalling this BHO is similar to installing any other Windows application and can be performed from the Add/Remove Programs from the Windows Control Panel. This plugin was tested to work in Internet Explorer 8 with Windows XP/7.

The current version has been presented to a team at Microsoft Research for review, but is not currently available for download.

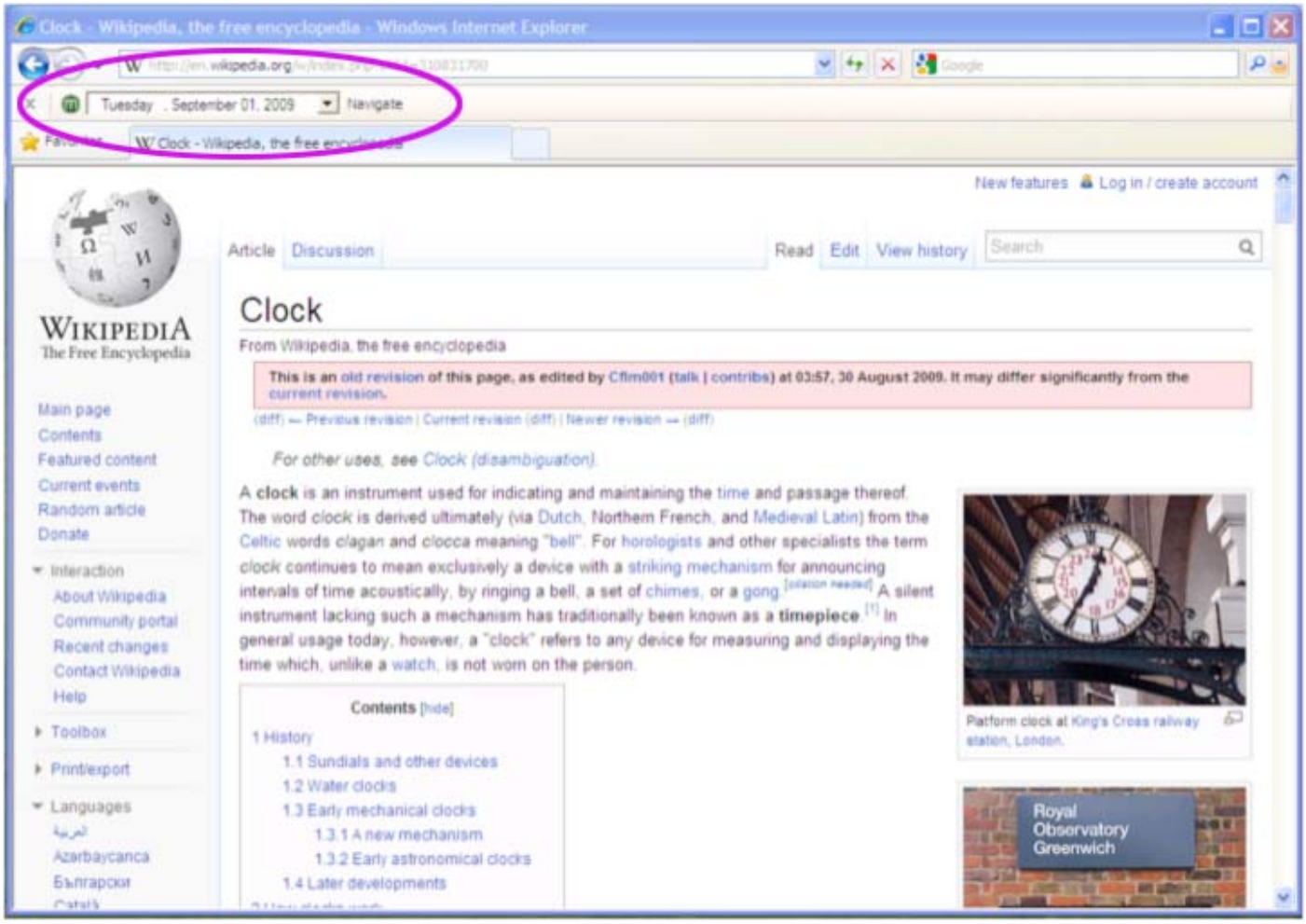

Figure 7: Screenshot of Memento BHO for IE

Android App: Memento Browser

To showcase Memento on a mobile platform, an Android application was developed called the Memento Browser. Android is a popular open source platform developed by Google and governed by the Open Handset Alliance. It is installed on a large number of smartphones from a variety of manufacturers. Applications are written in Java, and they can be downloaded from the Android Market or from any web page that exposes the application’s .apk file (a zip file composed of the executable code and other application files). Memento Browser was released as an open source project on Google Code in September 2010 [20].

A screenshot of Memento Browser is shown in Figure 8 below. In the top image, a user is browsing the mobile version of cnn.com on Sept. 16, 2010. To request a Memento for a different date, the user clicks on the button that is titled “2010/09/16”. In the center image, the user has requested a new date, July 7, 2009, and is presented with the nearest Memento as archived by WebCite on July 8, 2009. The left and right arrow buttons on the upper-right side of the browser allow the user to scroll through older and more recent Mementos. The user can also access a list of all known Mementos, as shown in the bottom image, by pressing the Android device’s Menu button and selecting “List Mementos”.


2:35 PM
http://m.cnn.com/
Go
Request:
2010/09/16
Showing:
2010/09/16
CNN
Weather
LATEST NEWS
AUDIO/VIDEO
Select a section
updated 03:28 PM EDT 09.16.10
CNN iReport


2:39 PM
http://webcitation.org/query
Go
Request:
2009/07/07
Showing:
2009/07/08
<
>
WebCite®
[What's this?]
Showing WebCite for
URL: http://m.cnn.com/
[Permalink to this cache]
Cached:
2009-07-08 12:17:38
CNN MOBILE .com
LATEST NEWS
Bomb parts smuggled into 10 federal buildings during test
Plainclothes investigators sent to test security...
• Paris Jackson emerges as surprise voice of memorial
• Goodbye Michael Jackson: Star, brother, friend, father
• Warwick: Jackson would've loved memorial service
See All News...
BLACK in AMERICA 2
WORLD

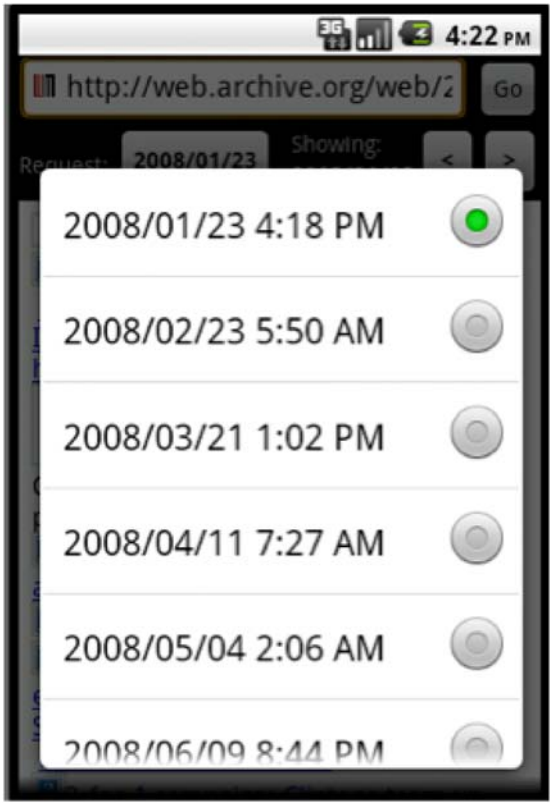

Figure 8: Screenshots of Memento Browser

There are a number of stock UI widgets that Android developers may use, including the WebView control which uses WebKit [21], an open source web browser engine, to render web pages. The WebView's `WebViewClient` provides the ability to override the web page's URL loading. The Memento Browser uses this mechanism in order to load the current version of a web page or a Memento from a requested date. However, the `WebView` control does not expose a mechanism to trap the HTTP requests and responses for the embedded resources of a web page, like the images, style sheets, etc. Therefore the current version of the browser does not use Memento for each individual element of a web page but instead relies on the archive that supplies the html to also supply each element via regular HTTP requests. When a user clicks on a link in a Memento, the Memento protocol is used again to find a web page that matches the requested date.

The Memento Browser uses threading extensively to perform HTTP requests without locking the UI thread. This allows the application to be responsive to user input while it is waiting for and processing HTTP responses. For example, when the user requests a specific date when browsing a web page, the browser starts a new thread which performs the Memento transactions as discussed earlier in Figure 1. When the browser discovers previous and next Mementos, it uses the Android Handler class to post a new Runnable to the UI thread's message queue. When this Runnable is executed, it enables the UI's previous and next buttons.

Memento Browser uses a default Memento TimeGate running at Los Alamos National Laboratory. However, a user may enter a different default TimeGate that may expose different Mementos. Currently the Memento Browser does not offer many of the frills that other browsers offer like, for example, bookmarking. However, Firefox launched a beta of their browser for Android in October 2010 [22]. As their browser stabilizes and becomes more popular, a new MementoFox add-on for the Android Firefox may offer a better alternative to a Memento-only web browser.

Future Strategies

The approach described above is tested and known to work when Memento is fully supported by the archive environment and by all browsers. However, an alternative strategy is being explored for environments in which this is not currently possible, such as the limited Android WebView control, and for the inevitable interim period in which Memento is not natively supported by browsers and thus the URIs must be rewritten for the archived pages to be viewable. It also supports a hybrid world in which some archives rewrite and some do not, as well as archives that rewrite some URIs but not others, such as an institutional archive that only rewrites internal URIs but leaves external ones pointing to the Original Resource.

The experimental setup allows the archive to optionally rewrite the embedded URLs, but for the client to be able to determine whether or not they have been rewritten. For limited-capability client platforms, this allows them to simply display the page as

transmitted, but at the same time allows full clients to determine if the performance increase in simply accepting the HTML outweighs the accuracy of using TimeGates to potentially discover resources that are closer to the requested timestamp.

The method being tested is for the archive to include a `Link` header, with a `rel` attribute of `original` on every response served, even if the response is an error condition or internal redirect. Combined with the `Memento-Datetime` header, served only on the actual Memento response, this allows the client to decide if it should accept the response or use the Original URI with a TimeGate to find a more appropriate representation. On the other hand, if the response does not include a link to the Original, then the client can conclude that the resource is the Original and the link has not been rewritten. In this case, the client should use a TimeGate to discover the appropriate Memento. A full classification of the different possible types of resource is available at the Memento site[23].

This setup would prevent the undesirable bifurcation of archives into those with rewritten pages and those with non-rewritten pages that can only be viewed with Memento clients, and the equally undesirable duplication of URIs to have two for each archived resource, one rewritten and one unrewritten. The archive can make the choice based on their own factors as to whether or not to rewrite, and Memento clients will be able to distinguish and process both scenarios. It also allows clients to optimize for speed by not requiring all of the HTTP requests and simply accepting “good enough” rather than as close to perfect as is available.

Conclusions

Development of the various Memento clients is ongoing. MementoFox is coordinated through the Google Code repository, and it is hoped that increased participation will occur as community engagement increases. It is also hoped that making the source code available will be encouraging to developers trying to implement the approach in other browsers and environments, such as Safari, Opera and Chrome. Some initial work has been performed on building a Memento Browser for the iPhone; however, it is not yet ready for publication.

More work is necessary on the user interface side as, although improvements have been made compared to previous versions, most of the effort so far has been in ensuring adherence to the Memento protocol. Acknowledging the archives that contributed resources to the view being displayed is an important user interface enhancement, although challenging to do without interfering with the rendering of the resources. Further customization to allow the user to define a period of time in which resources are acceptable, customization of searching across multiple TimeGates, and a way to better handle error conditions are also prominent in the issues list.

It is hoped that further Memento development will be inspired by sharing this technical information, and time and frustration will be saved through learning from our experiences. Access to the historical web has long been recognized as important; however, with the increased visibility of preservation of our digital cultural heritage, it has come to the forefront. Memento offers a transparent method for giving access to that heritage, which is currently hidden in archives around the world.

Acknowledgements

The authors would like to acknowledge the direct and indirect input of Herbert Van de Sompel, Michael Nelson and Luda Balakireva, our co-collaborators in the Memento project. We would also like to thank the Library of Congress for supporting the Memento research, and JISC for providing the prize for the Dev8D competition.

References

[1] RFC 2995: Transparent Content Negotiation in HTTP, March 1998 http://tools.ietf.org/html/rfc2295
[2] Tim Berners-Lee. Cool URIs don’t change. 1998 http://www.w3.org/Provider/Style/URI
[3] Memento – Adding Time to the Web http://www.mementoweb.org/
[4] Internet Archive – Wayback Machine http://web.archive.org/
[5] The National Archives http://www.nationalarchives.gov.uk/
[6] Internet Draft: HTTP framework for time-based access to resource states — Memento. v01 2010-11-11 http://www.mementoweb.org/guide/rfc/ID/
[7] RFC 5988: Web Linking, October 2010 http://tools.ietf.org/html/rfc5988
[8] RFC 1123: Requirements for Internet Hosts — Application and Support, October 1989 http://www.ietf.org/rfc/rfc1123.txt
[9] Open Archives Initiative – Object Reuse and Exchange http://www.openarchives.org/ore/
[10] Memento Web Application Demo, http://www.mementoweb.org/demo/client1/ (now defunct)
[11] Mozilla Firefox http://www.mozilla.com/
[12] Mozilla Firefox Add-Ons https://addons.mozilla.org/
[13] Mozilla Development Network Doc Center: Extensions https://developer.mozilla.org/en/Extensions
[14] JISC Developer Days, 2010 http://dev8d.org/dev8d-2010/
[15] Adblock Plus http://adblockplus.org/

[16] Mozilla Firefox Add-Ons: MementoFox 0.9.2, October 22, 2010 https://addons.mozilla.org/en-US/firefox/addon/100298/
[17] mementofox, Firefox Extension for Memento http://code.google.com/p/mementofox/
[18] HTML and CSS, Microsoft Developer Network Library http://msdn.microsoft.com/en-us/library/aa902517.aspx
[19] The Code Project: Extending Explorer with Band Objects using .NET and Windows Forms, 29 Apr 2002 http://www.codeproject.com/KB/shell/dotnetbandobjects.aspx
[20] memento-browser, A Web browser for Android http://code.google.com/p/memento-browser/
[21] The WebKit Open Source Project http://webkit.org/
[22] The Mozilla Blog: Firefox 4 Beta for Android and Maemo is Now Available, Posted by Stuart Parmenter, October 7th, 2010 http://blog.mozilla.com/blog/2010/10/07/firefox-4-beta-for-android-and-maemo
[23] Memento Guide: Determining Resource Type http://www.mementoweb.org/guide/resourcetype

About the Authors

Robert Sanderson (rsanderson@lanl.gov), is a Scientist in the Research Library at Los Alamos National Laboratory. His current research is focused on web scale interoperability for annotations (OAC) and for access to resource versions (Memento). He obtained his PhD from the University of Liverpool, where he was subsequently a Lecturer in Computer Science.

Harihar Shankar (harihar@lanl.gov), is a research and development engineer in the Prototyping team at the Research Library of Los Alamos National Laboratory, working on the Memento project. He has a Masters degree in Computer Engineering from the University of New Mexico.

Scott Ainsworth (sainswor@cs.odu.edu), has an M.S. in Computer Science and has worked in the field for 35 years. He is currently a Computer Science research assistant and Ph.D. student in the Web Science and Digital Libraries Group at Old Dominion University where he works on the Memento Project.

Frank McCown (fmccown@harding.edu), is an assistant professor of computer science at Harding University. His research interests include web archiving, web information retrieval, and mobile application development.

Sam Adams (sea36@cam.ac.uk), has a PhD in Chemical Informatics, and is currently a software developer at the Unilever Centre for Molecular Science Informatics, University of Cambridge, where he is working on a number of projects promoting the open publication of scientific data.

One Response to "Implementing Time Travel for the Web"

1. Open Access – do you really think it's a good idea? « digitalcollaboration, 2011-04-12

   […] Implementing time travel for the Web http://journal.code4lib.org/articles/4979 […]