AJAX Time Machine

Andy Brown and Simon Harper
School of Computer Science
University of Manchester
Manchester, UK
[andrew.brown-3 | simon.harper]@manchester.ac.uk

ABSTRACT

Many modern Web pages update parts of their content, and this is often automatic. This allows a 'clean' user-interface and information-rich pages. Keeping up with updates or recovering from mistakes can be a problem, however, as it is often not possible to revert a page to a previous state. This can be particularly problematic for users with poor literacy or cognitive disabilities, the elderly, or for users of assistive technologies. For pages that use these technologies to be truly accessible for all, they must afford users sufficient control over updates, to allow them to read and use the information available before it disappears forever. While applying good practice during page design and implementation can provide this, there are many pages for which information changes too rapidly for the user. We propose to supplement assistive technologies with a Web page 'time machine' that will allow users to review all the states a page has been in, and to step backwards or forwards through these states at their own pace.

Categories and Subject Descriptors

H.1.2 [**Models and Principles**]: User/ Machine Systems—*human factors, human information processing*; H.5.4 [**Information Interfaces and Presentation**]: Hypertext/ Hypermedia—*user issues, navigation*; K.4.2 [**Computers and Society**]: Social Issues—*assistive technologies for persons with disabilities*

General Terms

Human Factors

Keywords

Web 2.0, AJAX

1. INTRODUCTION

Updating pages are now a well-established feature of the Web. Users are expected to cope with them, indeed embrace them and benefit from them, and many do. For others, however, these features have the potential to be confusing and make it difficult for them to complete their tasks.

Chadwick-Dias *et al* examined the issues faced by older users with Web 2.0 [1]. One of the problems they identified was that "there is much inconsistency in how links and other elements look and act", and older users "often had no idea what is actionable on a page". They also found problems with the controls embedded in Web 2.0 pages: "many widgets have small controls (arrows, +/−, etc.) that are often very difficult to see and with which to interact", and noted a general hesitancy about the way in which these people used the Web. Hesitancy was also noted in older users by Lunn and Harper [3], and is indeed a behaviour typical of older people [4], who also show difficulty in maintaining attention, focus, and concentration on tasks where there is a lot of distracting information [2].

Zajicek looked at the potential impact of Web 2.0 on accessibility in general [6]. While many of the problems she identified with AJAX are being improved with improvements in screen readers and the use of WAI-ARIA markup, these clearly remain an issue for many users, particularly those who do not have the money, time, or know-how to keep upgrading to the latest software. She also identifies the problem of remembering new features, a problem which relates to both the functionality of the pages themselves and the additional commands required to handle dynamic content with a screen reader.

The issues of identifying and using controls correctly applies not only to older users, but to many others, including those using screen readers, and people with cognitive or motor impairments [5]. It is clearly difficult for many people to use Web applications, and it is unsurprising that their behaviour has been described as hesitant. We propose that part of the reason for this is an inability (or at least a perceived inability) to undo many actions. Currently, a wrongly-remembered action or an input error can lead to changes to a page that are difficult, or perhaps impossible, to undo. Problems with automatic updates can also be identified — in these cases, sections of a page change content without any input from the user, e.g., to give a running commentary on a sports fixture. In this situation, it is likely that users who cannot process the information in sufficient time will not benefit fully from the page.

We may now list several ways in which different users may benefit from the ability to undo or review changes to a dynamic Web page:

- The ability to undo the effects of remembering and using

W4A2011 - Communication, March 28-29, 2011, Hyderabad, India. Co-Located with the 20th International World Wide Web Conference.

the wrong action, or activating a control by mistake.
- The ability to follow a series of automatic changes at a pace set by the user.
- Providing accessible notification of changes.
- A potential decrease in users' hesitance.

In this paper we present a brief analysis the problem and describe an implementation we are in the process of building. First we consider what the requirements of such a system are: what must it be able to do to help a range of users cope with dynamic content (section 2)? Second, we describe our prototype, giving an overview of its design and current capabilities (section 3), and finally outline our plans for development (section 4).

2. REQUIREMENTS

From the understanding of the problems faced by users and the potential benefits offered by an undo/replay system for dynamic pages, it is possible to enumerate a set of requirements. Such a system must:

1. Not interfere with normal page operation.
2. Keep the user informed of the status of the page or region: have there been any updates, and if so, how many?
3. Handle updates as perceived by the user, i.e., at a high-level.
4. Record all the content, including that input by the user.
5. Enable output in different forms, to suit different types of user.

In addition, we note that it is also highly desirable that the system can retain interactivity during replay (i.e., fully revert to a previous state). However, several issues make this a difficult proposition, and a system that does not have this ability could still benefit users.

3. PROTOTYPE

Our implementation, known as the AJAX Time Machine, is a prototype of a system that we intend to fulfil many or all of the requirements outlined above. It currently takes the form of an extension to the Google Chrome browser, and its primary aim is currently to test the mechanisms for recording and replaying updates. As such, its rendering is aimed only at sighted users, and its user-interface is a simple one using the mouse.

In outline, the AJAX Time Machine works by enabling users to select regions of a page for update recording. When this is done, a small visual user-interface is injected into the page. This contains links that control the content of the region, showing how many updates have occurred and allowing users to step backwards and forwards through them. A browser toolbar button also opens a popup showing a summary of all updates on a page — Figure 1 shows the updates that occurred as a result of clicking the 'News' tab on the Yahoo! UK home page.

3.1 User Interface

Despite the fact that the user interface is not yet optimised for those who would most benefit from such technology, describing it will make the explanations of the techniques used for recording and replaying updates clearer. The GUI region (shown in Figure 2) contains 4 sections:

- A left arrow. Clicking on this changes the region content back one state (unless viewing the original version).

Figure 1: The update summary popup: the user has clicked the News tab on the Yahoo home page.

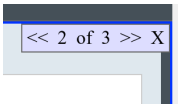


Figure 2: The GUI of the prototype. The user is viewing the page as it was after the second of three updates.

- Context information. The total number of states that have occurred is displayed, together with the number of the one currently being displayed. State 1 is the region as it was when the user started recording.
- A right arrow. Clicking on this changes the region content forward one state (unless viewing the most recent version).
- Close button. Clicking this resets the region to its most recent state, stops recording and removes the user interface.

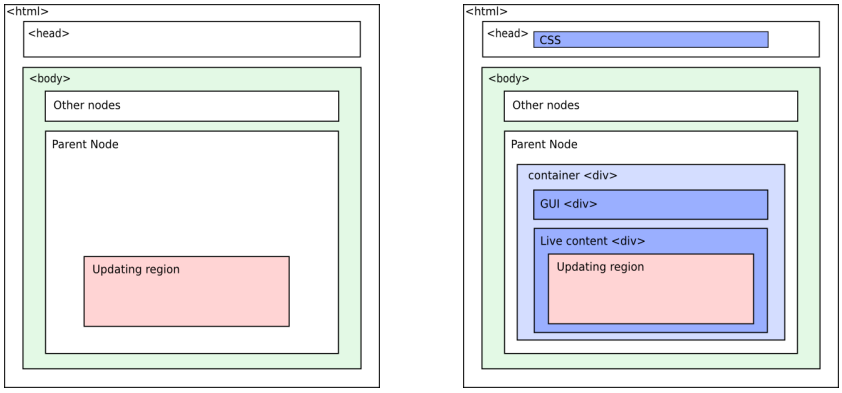
Scrolling the mouse wheel while the pointer is over the GUI region will also move the user backwards and forwards through the page history.

3.2 Initiation

Although we are investigating the possibility of automatically identifying and recording updating regions, the AJAX Time Machine prototype currently works by allowing users to specify regions to record. This is done using a right click on the page, upon which the region clicked on is highlighted briefly, and the user is given the option to add it as a live region. If the user opts to do this, then a new HTML `<div>` element (the 'container `<div>`') is inserted into the DOM tree between the live region node and its parent (see figure 3). This is styled with a border, to show the user which region has been selected, and contains not only the content (within its own `<div>` — the 'live content `<div>`'), but also a GUI `<div>` (as described in section 3.1). Once the DOM has been suitably modified, the current content of the region is recorded. If the close button on the region user interface is activated, the DOM tree reverts to its original structure, but with the latest content. Styling is achieved with CSS injected into the `<head>` element.

3.3 Recording Updates

Updates are detected using the DOMMutationEventListeners: Listeners for DOMNodeInserted, DOMNodeRemoved



(a) Initial DOM (b) DOM after GUI insertion

Figure 3: DOM modification to show a user interface, and enable manipulation of the updating content element.

and DOMCharacterDataModified events are added to the document. Each time an update occurs it is handled by these listeners, which test if the changes have occurred in any of the recording regions and test if the changes are meaningful. Meaningful updates are considered to be those where a typical user would identify a change, so do not include modifications to comment nodes, hidden nodes (i.e., parts of the document which are styled to be invisible, such as `style=''display: none''`), changes in white space, or updates that do not actually change the content (e.g., a regularly updating sports score, where the score does not necessarily change for each update). The content is recorded by simply storing the HTML and the input values. A short delay between receiving the DOMMutationEvent and recording the event helps group the low-level updates into changes as they appear to the user (see section 4.1.2).

3.4 Replaying Updates

Replaying is controlled by the GUI, as described in section 3.1. Currently, the AJAX Time Machine displays old states of a page region by simply replacing the content of the 'content `<div>`' with the HTML and input area values recorded for a given state. Thus, the user is able to view the original state in the context of the whole page. Note that if there are multiple updating regions selected on the page, the user replays them independently.

The AJAX Time Machine continues to monitor for updates while the user is browsing. This is potentially valuable for automatic updates where the user needs to read a whole sequence but, for example, cannot read quickly enough to keep up to date. In these cases the new content is recorded, while the region continues to display the state specified by the user. This requires some careful design, as content replacement caused by browsing the history triggers DOMMutationEvents, which have to be distinguished from those that are caused by genuine updates.

4. DISCUSSION AND FUTURE WORK

This prototype demonstrates that it it technically feasible to record a page as it updates, and allow the user to review the changes. There remain many challenges, however, both technical and relating to presentation, in particular how the recordings may be presented best to users who will benefit most. In this section, we discuss these problems, and propose some ideas that should improve the accessibility of this type of information.

4.1 Technical and Design Challenges

4.1.1 Defining regions to record

Manual selection of a region is a surprisingly difficult task, as users must click on an area of the page that contains the target region, but not on any of its children. A second issue with manual selection is that it will not capture unexpected updates, which are probably those that will most need to be reviewed. There are two alternative solutions to these problems: record the entire page, or automatically detect and record individual regions. The former has the advantage of recording changes in context, e.g., if updates are inter-related, and simpler user interaction (only one set of controls), while the latter allows for situations where users may want to review one section of a page while editing (e.g., completing a form) another. Disadvantages of these approaches include more demand on the computer (memory and processor), which may prove problematic on complex pages and/or for users with older hardware. This may be reduced for the approach of recording regions by asking the user if a region should be recorded, but this comes at the expense of considerable interruption to the user.

4.1.2 Chunking Updates

'Chunking' is the term we use to describe the process of combining low-level updates into coherent units, as would be perceived by users. The problem arises because DOMMutationEvents are generated for each leaf node in the DOM, while these are usually invisible (and irrelevant) to users. For example, removal of a section of page generates events for each paragraph removed and for each white space node between paragraphs, while the user perceives the section vanishing as a single event. This poses difficulties when developing a tool such as the AJAX Time Machine, as users will want to replay the changes as they perceived them, not in micro-steps as detected by the DOM, so it is necessary to group the changes.

The only information that is available to do this is the time of the update and the region of the tree it affected.

The approach in the prototype is to pause briefly (currently 100ms) after an event is detected before recording. This allows related events to occur and captured as one chunk; the remaining low-level events from the change are ignored as no changes are detected. This approach is effective for DOMNodeRemoved events, as these occur in close succession, but DOMNodeInsertion events may be delayed by factors such as download time, so a more effective system may be necessary.

4.1.3 Presentation

One further technical challenge is to present the user with controls that are clear, understandable, and usable, while not interfering with the underlying page. The current implementation, with a small GUI inserted into the page is relatively clear in how it should be used, but the simple CSS used to generate it does not always display it correctly on 'real-world' pages. It also has the potential to obscure important information. Development of a more robust user interface is necessary. It should be noted that this problem is simplified if the whole page is recorded (see the discussion in section 4.1.1), as a single set of controls will suffice, and may be located off the page.

4.1.4 Reverting Changes

While many updates are simple changes to the DOM, others involve changes that are difficult to identify or revert, e.g,. information may have been updated on the server. Although reverting to an earlier state of the page that is fully interactive is not likely to be achievable, it may be possible to identify when this is the case, and inform users that it is only possible to review earlier states.

4.2 Supporting Users

The premise behind this development is that some users find it difficult to use pages with dynamic content, and that the ability to review or return to previous states of the page would be useful. As discussed in the introduction, those users most in need of this type of assistance are likely to be those with other needs, such as users of screen-readers or screen magnification tools, cognitively impaired users or the elderly. A tool such as this must therefore support these users in the way it presents information and in the design of its user interface. The key aims are to:

- Make it clear that updates have occurred.
- Enable users to review previous states of a page.
- Make previous states fully interactive, i.e., behave as if they had never left that state.

4.2.1 Apply WAI-ARIA markup

WAI-ARIA is a collection of markup from the World Wide Web Consortium (W3C) Web Accessibility Initiative (WAI) that can be applied to Web applications to make them more accessible. The principle is that the markup makes the behaviour of the application explicit, where it might otherwise be implicit. For example, standard HTML components may be made to behave like a tree using styling and scripting, and while this may be obvious when presented as intended it is not otherwise (e.g., in audio); WAI-ARIA gives developers a method for making the role of the components explicit. Applying ARIA markup to the injected code will make it easier for screen reader users to understand when changes have occurred (on pages that don't use ARIA), as well as to control and explore the replays.

4.2.2 Keyboard controls

Although the prototype UI is mouse-only, keyboard control is essential for many types of user. This is simple to achieve, but complications may arise. For example, rich internet applications will typically have keyboard controls built in, and using simple, consistent controls for the AJAX Time Machine, while avoiding conflict with pages may prove difficult.

4.2.3 Presentation

While the current implementation replays updates by simply replacing the current state of a region with a previous state. Understanding updates with a more narrative structure, however, might be supported better by showing all versions of a section together. Additional support could be provided by making the changed parts of a region more explicit using some form of highlighting.

5. CONCLUSION

Many different types of users can potentially benefit from the ability to undo actions on a page, or review previous states. We have built a prototype that demonstrates the idea, and shows that it is feasible. We have also identified several areas for development — moving the implementation to record the page as a whole appears to be necessary. Further work is required to implement these, and to undertake user testing.

6. ACKNOWLEDGEMENTS

This work is funded by a Google research award.

7. REFERENCES

[1] A. Chadwick-Dias, M. Bergel, and T. Tullis. Senior surfers 2.0: A re-examination of the older web user and the dynamic web. In C. Stephanidis, editor, *Universal Acess in Human Computer Interaction. Coping with Diversity*, volume 4554 of *Lecture Notes in Computer Science*, pages 868–876. Springer Berlin / Heidelberg, 2007.

[2] C. J. Ketcham and G. E. Stelmach. Age Related Declines in Motor Control. In J. E. Birren and K. W. Schaie, editors, *The Handbook of Psychology and Aging*, chapter 13, pages 267–287. Academic Press Inc, London, UK, 5th edition, 2001.

[3] D. Lunn and S. Harper. Using Galvanic Skin Response Measures to Identify Areas of Frustration for Older Web 2.0 Users. In *W4A '10: Proceedings of the 2010 International Cross Disciplinary Conference on Web Accessibility*, pages 1–10, New York, NY, USA, May 2010. ACM.

[4] D. Memmert. The Effects of Eye Movements, Age, and Expertise on Inattentional Blindness. *Consciousness and Cognition*, 15(3):620–627, 2006.

[5] S. Trewin, S. Keates, and K. Moffatt. Developing steady clicks: a method of cursor assistance for people with motor impairments. In *Assets '06: Proceedings of the 8th international ACM SIGACCESS conference on Computers and accessibility*, pages 26–33, 2006.

[6] M. Zajicek. Web 2.0: hype or happiness? In *Proceedings of the 2007 international cross-disciplinary conference on Web accessibility (W4A)*, W4A '07, pages 35–39, New York, NY, USA, 2007. ACM.