SASWAT Technical Requirements

Andy Brown and Caroline Jay
Human Centred Web Lab
School of Computer Science
University of Manchester
UK

The range and scope of dynamic updates have been introduced in the technical review document, and a framework for making these updates more accessible has been developed based upon eye-tracking studies and a review of the current state of assistive technologies. This report considers the findings of all this previous work to give the technical requirements for a system that will make the proposed mappings. These requirements will form the basis for a tool that allows these mappings to be evaluated, and suggest that this tool will be a modified version of the Fire Vox screen reading extension to the Firefox Web browser.
SASWAT

The aim of the SASWAT project is to develop a framework for mapping the competing dynamic micro content produced by Web 2.0 technologies to audio. The SASWAT web pages may be found at http://hcw.cs.manchester.ac.uk/research/saswat/.

SASWAT Reports

This report is in the series of HCW SASWAT technical reports. Other reports in this series may be found in our data repository, at http://hcw-eprints.cs.man.ac.uk/view/subjects/saswat.html. Reports from other Human Centred Web projects are also available at http://hcw-eprints.cs.manchester.ac.uk/.

Acknowledgements

This report forms Deliverable 4, Technical Requirements, for the SASWAT project. SASWAT is funded by the EPSRC, reference: EP/E062954/1.
Contents

1 Introduction 1

2 Requirements 1
  2.1 Monitoring .............................. 1
  2.2 Characterising ............................. 3
  2.3 Scheduling ................................. 3
  2.4 Presenting .................................. 3
  2.5 Summary of Requirements .................... 4

3 Implementation 4
  3.1 Orca ........................................ 5
  3.2 Fire Vox ..................................... 6
  3.3 Evaluation .................................... 6
  3.4 Conclusions ................................... 7

4 Summary 7

Human Centred Web Lab
School of Computer Science
University of Manchester
Kilburn Building
Oxford Road
Manchester
M13 9PL
UK
tel: +44 161 275 7821
http://hcw.cs.manchester.ac.uk/

Corresponding author:
Andy Brown
tel: +44 (161) 275 7821
andrew.brown-3@manchester.ac.uk
http://homepages.cs.manchester.ac.uk/~browna
1 Introduction

The aim of the SASWAT project is to build an understanding of how dynamic updates, which were designed to function in the visual medium, might be transformed so as to be effective when used in an audio environment. The ways in which sighted users allocate attention when interacting with these updates have been studied with eye-tracking studies [3], and will form the basis for a model of how updates might be transformed. This report examines the technical requirements for implementing a system that can be used to test if these mappings make dynamic updates more accessible. Since development of the model is currently in progress, the requirements here are given at the level of what the system needs to be capable of (rather than how each type of update should be dealt with). It is based upon the findings of the initial studies, coupled with the technical analysis of how updates may be achieved [2]. The requirements set out below are designed to ensure sufficient flexibility to allow all likely outcomes of the model to be implemented, and to allow improvements as iterative evaluation proceeds.

2 Requirements

Any system that is designed to test the model must be able to collate all the information required by the model, and be sufficiently flexible to present updates at any time, and in different ways. The requirements must, therefore, specify what information may be needed to make the decisions about presentation, and the different ways in which the update may be presented.

The key outputs of the model are predictions of when to present an update (possibly never) and what to present. The aim of the evaluation is to test whether the outputs of our model enable efficient use of the updates. The iterative evaluation planned means that the implementation must be able to cope with an evolving model. Thus, we need to be able to:

1. Monitor the page and the system as a whole, so that changes to the page (i.e., updates) are identified, and the user’s state (his or her current activity, preference settings, etc.) is known.

2. Characterise any updates: identify those attributes of an update that the model predicts are important for determining when and how to present it.

3. Schedule updates according to their characteristics.

4. Present updates according to their characteristics.

Each of these is discussed in more detail below.

2.1 Monitoring

Several things need to be monitored during browsing; most importantly, it is necessary to detect when and how a page updates. ‘Dynamic update’ is a relatively loosely defined term, but for the purposes of the evaluation phase of this project,
dynamic updates are taken to be any change to the document that occurs without reloading the whole page. A change to the document is anything which results in a change to the Document Object Model. While this does not capture all types of dynamically changing Web pages (e.g., those using Flash or Java applets), it should be sufficiently general to allow the model to be evaluated. Thus, the most important requirement is the capability to monitor the page’s Document Object Model (DOM).

While the focus of this research is dynamically updating micro-content, an extended approach has been proposed (and implemented) by Borodin et al. [1] where the DOM’s are compared both within and between pages. For example, a page may have a sizeable heading and menu structure, where following a menu link loads a new page with the same structure, but different central content. Borodin’s approach allows all types of change to be considered in the same way, so the example above may be treated as a dynamic change of the central content, removing the need for the user to move through all headings and menus to discover the new central content. While implementing a system such as this is not a requirement, it may be desirable, should it prove simple to achieve.

A second aspect of monitoring to consider is that it will probably also be important to able to store and retrieve the preferences of an individual user. One can envisage a situation where the user decides that he or she has no interest in the information provided by a particular update, and therefore wants to ignore all future updates. Conversely, some information might be particularly interesting, and the user may wish to be informed immediately if it is updated. Thus, the system should allow users to set preferences, not only regarding the behaviour of the system as a whole, but also for individual updates.

A further piece of information that needs to be known is where the user is currently allocating his or her attention. Since information is being given aurally, through the screen reader, it is possible to track what the user is currently reading. This information might prove useful when considering precisely when to present a particular update. For example, while an important update might be presented as soon as it occurs, a less important one might be presented when the user has finished listening to the sentence currently being spoken.

Reading a Web page through synthetic speech is more time-consuming than reading it visually. Anecdotal evidence suggests that screen readers that present updates immediately can become annoying when updates happen frequently. Users complain that their focus of attention is moved to the latest update before they have finished reading the previous one. Similar situations might occur when content is replaced automatically. While scheduling updates according to our model might alleviate this problem, in order to allow users to fully explore the information on a page, it might prove useful to allow them to return to previous states. Monitoring the system might need to include tracking the state of the document, to make these previous states accessible.
2.2 Characterising

The following update characteristics were identified as being factors in whether users allocate attention to the new information:

- The initiating event. Was the update triggered automatically, or by some user activity?
- The user’s action. If an update was user-initiated, what action caused it — button or link click, hover. Since mice are rarely used with screen readers, it will be necessary to know by what action the update was intended to be triggered, and by what action it was triggered.
- How has the page changed. What effect has the update had on the page — has information been added, removed, replaced, or rearranged?
- What information is provided by the update.

2.3 Scheduling

From the model, scheduling will be based upon the information gathered from general monitoring and characterisation of a particular update. Thus, presentation time may depend upon:

- What other updates are pending.
- The preference settings.
- The characteristics of the update.
- The user’s current activity (e.g., is he or she mid-sentence, reading another section entirely, or waiting for a new page to load?).

2.4 Presenting

The fundamental aim of this research is to make dynamic content truly accessible, i.e., process and present it in a way that allows visually impaired users to read both the host document and the updates in an effective and efficient manner. The balance between these can be considered as like a conversation: the updates need to be scheduled and presented in a manner that allows a natural flow of information. To achieve this, new information needs to be introduced both at an appropriate time and with an appropriate level of detail.

While the model predicts how likely it is for a user to allocate his or her attention to an update, it also predicts what parts of the information in that update are likely to be read. This shows that it is not always appropriate to provide the whole update — for example, it was unusual for more than the first three suggestions of an auto-suggest list to be fixated. It is therefore necessary for the way in which an update is presented to be flexible. For information that appears more complex, it is important to take into account the difficulties associated with presenting through
sound, in particular how difficult it is for readers to get a glance or overview of the information. We might therefore suppose that it will be useful to replicate this facility by providing a summary of an update.

Although the user-studies showed that there were many instances where dynamically updating content was completely ignored by participants, it would not be reasonable to discard these updates entirely. While it might be appropriate to allow the update to occur unannounced, it could be useful to use a simple tone to notify the user; he or she may then investigate its content at a convenient time. This will require an implementation that can store updates, summarise them, and enable either direct or delayed access to them.

2.5 Summary of Requirements

Combining the analysis of the four needs, above, we can identify the following specific requirements. The system must be able to:

1. Monitor and track the DOM
2. Characterise any changes to the DOM, in terms of:
   (a) Initiating event
   (b) Change of page structure
   (c) Semantics of new information
   (d) Presentation of new information
3. Hold updates in a queue, and modify that queue
4. Summarise content and changes
5. Monitor the user’s focus of attention (screen reader activity)
6. Store and modify user preferences (global and update-specific)

Section 3 discusses the implications of this for implementation, with reference to Deliverable 3 [2], which discusses the techniques developers can use to achieve dynamic updates.

3 Implementation

What are the implications of these requirements for implementation? What form does the prototype need to take so that it can fulfil the requirements above? While the proposal for this project envisaged ‘a middleware tool or software bridge to map parallel competing Web-2.0 visual outputs to a harmonious serial audio output’, the requirements suggest that the solution will need to be more tightly integrated with the screen reader.

Monitoring and characterising requirements demand access to the page source code (including, probably, JavaScript, so that the cause of updates may be determined) and document object model (DOM). The monitoring and presenting requirements also suggest that it will be necessary not only to monitor the output of the
screen reader, but also to control when certain items are to be spoken. Meeting these requirements is only achievable if the way in which the screen reader works can be modified, i.e., it will be necessary to modify the source code of a screen reader. It is not clear at this stage whether access to the screen reader code will be sufficient, or whether the browser will need to be queried or even modified too.

Open source screen readers include Orca\(^1\), on the Gnome\(^2\) desktop for Linux, and NVDA\(^3\) on Windows, while Fire Vox\(^4\) is an extension that turns Firefox into a self-voicing application, and works on Linux, Windows and Apple OS-X. Firefox is the clear choice of open-source browser, indeed the only browser with significant support in Orca, NVDA or Fire Vox. Since NVDA does not appear to offer any significant advantages, it was decided that, for the convenience of development (HCW Lab computers are largely Apple OS-X or Linux), only Orca and Fire Vox would be investigated more deeply.

### 3.1 Orca

Orca is a full screen reader for Gnome desktop. Accessibility on Gnome\(^5\) is provided by the Accessibility Toolkit (ATK — interfaces that allow GUI components to be accessible) and the Assistive Technology Service Provider Interface (AT-SPI — an API that allows assistive technologies to obtain accessibility information from running applications). Figure 1 gives an overview of the GNOME accessibility architecture.

---

\(^1\)http://live.gnome.org/Orca  
\(^2\)http://www.gnome.org/  
\(^3\)http://www.nvda-project.org/  
\(^4\)http://www.firevox.clcworld.net/  
\(^5\)http://library.gnome.org/devel/accessibility-devel-guide/nightly/
While this architecture is suited to the heterogeneous environment of a desktop, it has some limitations, which might make it difficult to access the necessary information. For example, Orca can access the accessible DOM, which is not the whole document content. According to Mozilla:

“The accessible tree and the DOM tree are parallel structures. Accessible objects are created in the accessible tree for every DOM element that should be exposed to AT. Roughly speaking the accessible tree is a subset of the DOM tree. The following rules are applied to define whether DOM element is exposed to AT.”

It is therefore possible that, to fulfil the requirements, modification would be necessary not only to the Orca code (implemented in Python), but also to the source code of the AT-SPI interface and Firefox (C and C++).

3.2 Fire Vox

Fire Vox is a Firefox extension, so runs directly within Firefox, rather than through an intermediate interface. Perhaps as a result of this, accessing the content, and any changes to it appears easier than with Orca. Firefox extensions are written in JavaScript, and have access to the same DOM as any inline page scripts, so can be used to read and modify the DOM (requirements 2b, 2c, 2d and 4). It also allows capturing DOM Mutation Events, allowing it to detect when the page has changed (requirement 1), and other events (e.g., keyboard and mouse), allowing the user’s activity to be monitored (for requirements 2a and 5). The text to speech interface used in Fire Vox, allows a certain amount of tracking what is spoken (requirement 5). Fire Vox has its own set of user preferences, which could be extended to allow requirement 6 to be fulfilled (indeed, updates marked with ARIA tags may be turned off by the user already). Requirement 3 will require implementation, but is clearly achievable.

Fire Vox has a much simple interface than Orca, with fewer commands, but also fewer facilities. For example, it is not easy to jump from heading to heading, as it is in Orca. While this should make it easier for participants to learn to use, it might detract from their browsing experience. It may be necessary to implement some extra functionality in Fire Vox. The differences largely hinge on the fact that Fire Vox uses the Firefox caret when browsing, while Orca allows the user to switch caret control between Orca (recommended) and Firefox.

3.3 Evaluation

In order to allow an effective evaluation, it is necessary that the predictions that the model makes about the best way to present updates are tested. The ideal would be for participants to use their normal system, i.e., their usual browser and screen reader. For UK-based participants, at least, this would probably be JAWS and Internet Explorer [2]. Unfortunately, both JAWS and IE are closed-source,
so it is not possible to modify their behaviour. Furthermore, individuals configure
screen readers in different ways, so that even a modified version of JAWS could
vary significantly from the user’s normal experience. While the need to adapt an
open-source screen reader means either recruiting from a small user-base (those who
are already familiar with that software), or having participants using unfamiliar
software (the low uptake of open source technologies among the blind and visually
impaired community would at least mean that probably all participants would be in
the same situation), it is possibly the only way in which we have sufficient control
over the independent variables.

The user-base for NVDA is possibly larger than for Orca or Fire Vox, but not
sufficiently large to provide suitable numbers of experienced participants for an eval-
uation. Fire Vox has the advantage over Orca that it is a Firefox extension that
can be sent to users for qualitative feedback or pre-evaluation familiarisation (al-
though there may be issues with using a self-voicing application within a screen
reader environment). Fire Vox is also cross-platform.

3.4 Conclusions

While none of the available options are without disadvantages, it appears that Fire
Vox will be the easiest to modify, and the most certain to fulfil all of the requirements.

4 Summary

Web 2.0 technology allows Web pages to change dynamically, with updates of micro-
content rather than an entire page. These changes may occur automatically or as
a result of the user’s interaction with the page. They may affect the page in a
variety of ways, by adding, removing, replacing or rearranging content. The most
common technology used to achieve dynamic updates is XMLHttpRequest, but other
techniques may be used to achieve the same result. Monitoring the DOM of a page
for changes is the most promising candidate for detecting updates, as it should
be reliable, and it enables access to more semantics than the raw HTTP traffic.
Monitoring for these updates, characterising them, scheduling them, and presenting
them to the user will require modifications to the way in which the screen reader
works. Implementation will therefore involve changing the code of an open source
screen reader, so that it works in a way that is consistent with how our model
predicts sighted users allocate attention to updates.

Fire Vox is the implementation that appears to offer the best balance of capa-
bilities with development time. It should be possible to fulfil all the requirements
(section 2.5), while the tightly coupled nature of this extension with the browser
simplifies development in a way that will be beneficial when it comes to iterative
evaluation.

References

What’s new? — making web page updates accessible. In Assets ’08: Pro-
