AxsJAX: A Talking Translation Bot Using Google IM

Bringing Web-2.0 Applications To Life

Charles L. Chen
Google Inc.
clchen@google.com

T. V. Raman
Google Inc.
raman@google.com

ABSTRACT

Web-2.0 applications turn static Web documents into dynamic user interfaces. They epitomize the final realization of the vision "The Document **Is** The Interface!". This transition from static Web pages to interactive Web applications also requires the introduction of a fresh set of innovations to how such applications are accessed in conjunction with adaptive technologies.

Asynchronous JavaScript and XML (AJAX) breathes life into static Web pages. ARIA live regions helps bring such interaction to life when used in conjunction with adaptive technologies such as screenreaders and self-voicing browsers. This paper introduces the motivation behind live regions in ARIA, and describes how this support can be used to enhance the user interaction provided by Google Talk —an instant-messaging client that is integrated into the GMail Web interface. We describe the interaction model as it is surfaced to the end-user, and show how the introduction of *live regions* makes all aspects of the resulting UI usable with adaptive technologies.

Web-2.0 applications —especially mashups —excel at creating end-user solutions that are greater than the sum of their individual building blocks. We demonstrate this by bringing together Google Talk, Live Regions and Natural Language translation by demonstrating a multi-lingual talking translation interface that is the result of speech-enabling these applications using the Google AxsJAX framework.

1. INTRODUCTION

Web applications turn static web documents into highly reactive user interfaces. In this model of interaction, Web pages behave like traditional desktop applications —the Web page is the user interface canvas, and portions of this canvas update as a result of user interaction.

Enabling adaptive technologies like screenreaders and self-voicing browsers work with such applications requires the following:

Reflection The ability to discover the *role* and *state* of interaction widgets.

Notification The ability to detect *relevant* changes to portions of the user interface so that the user can be appropriately notified.

Reflection *i.e.,* the ability to discover the role and state of an interaction widget —brings Web-2.0 applications on par with traditional desktop applications when using a screenreader. Support for notification via live regions goes one step further by enabling rich auditory user interfaces to Web applications in a form that has been traditionally difficult to achieve —[3].

Reflection

Web applications can be augmented with reflection using ARIA [5, 2] properties `role` and `state`. When Web applications that have been annotated with these properties are rendered by a supporting browser, UI widgets implemented via HTML and JavaScript behave like native platform widgets when seen by the adaptive technology. As a consequence, interaction widgets constructed out of Web parts including HTML, CSS and JavaScript raise the same platform-specific accessibility events raised by native widgets.

Notification

Reflection as described above is one piece of the accessibility puzzle, and makes leaf-level widgets such as menus and sliders usable in the presence of adaptive technologies. However, a Web application is more than the sum of the individual widgets used to construct the user interface. To work effectively with such applications, users need to be notified about relevant page updates, and this involves identifying and automatically speaking portions of the Web page.

Notice that in the above, the task of *identifying* the relevant portion to speak is as important as the speaking of that information. To see this, consider a Web application that displays a continuously updating time display at the top, along with a dynamically updated set of responses that match a user query. This application has two *regions* that change —the time display, and the query responses. It is important to distinguish these two updates from the perspective of adaptive technology.

Having the adaptive technology automatically speak *all* page updates as they happen would result in an unusable application. At the same time, it is important to speak the updated results as soon as they are available while the user is interacting with that application.

W4A2008 - Communication April 21–22, 2008, Beijing, China. Co-Located with the 17th International World Wide Web Conference.

This form of timely and appropriate notification is enabled via *live regions* in ARIA [1]. Web application authors identify those portions of the Web page that need to be communicated to the user upon partial updates; they also declare the *priority* of such updates. With the appropriate live region properties in place, supporting Web browsers can then trigger the relevant feedback through conforming adaptive technologies during user interaction.

2. USER INTERACTION MODEL

Background

At the 2007 W4A conference, Peter Thiessen et al described a prototype chat client that they implemented to demonstrate ARIA live regions [6]. Drawing on that work, we have now added ARIA support as described in that paper to Google Talk IM, a mainstream instant-messaging client from Google that is fully integrated with GMail. The rest of this paper focuses on those aspects of speech-enabling Google Talk that are new with respect to what was described last year. In particular, subsequent sections of this paper describe how we speech-enable the integrated user experience brought about by the coming together of instant messaging and machine translation, and the resulting *talking interpreter* that materializes as a consequence.

The GMail User Interaction Model For IM

GMail uses a common contact list for mail and instant messaging. The Web interface shows a dynamically updated display of online contacts, and provides a shortcut key (`q`) for launching a chat session with an online buddy.

Invoking a new chat session brings up a simple user interface for a two-way text conversation. This display updates whenever a new message is received as part of a given conversation. The user can have multiple parallel conversations at any given time.

Initiating a new chat session via shortcut key `q` prompts for the chat buddy using an input field with auto-completion. The user can type a few characters and then use the `up/down` arrow keys to navigate through the available completions. The user interaction presented by these controls is speech-enabled using W3C ARIA —but we will not cover those details here since they have already been described elsewhere see [6].

The next section describes how we use ARIA properties `role` and `state` along with *live regions* to create a seamless user experience when using the keyboard along with spoken output. In this case, our task was made significantly easier by the availability of full keyboard access to the underlying features of GMail —note that though most actions in GMail are commonly performed via the mouse, that application provides a wide range of shortcut keys for the power user.

3. AUDITORY USER INTERFACE

This section sketches out the auditory user interface ([4]) for the IM application by enumerating the various user actions and the desired auditory feedback —we will describe the final implementation that achieves this in 4.

Notice that the asynchronous nature of instant messaging means that we need to proactively notify the user in the following instances:

- Incoming chat invitations.
- Arrival of new messages in an ongoing chat session.
- Subscription requests from friends wishing to *subscribe* to the user's presence.

Action	`Key`	Effect	*Feedback*
Launch	`q`	Open chat	*Opening chat*
Buddy	...	Auto-complete	*Speak completion*
Confirm	`RET`	Create chat	*Chatting with* buddy
Send	...	Type message	*Echo message*
Receive		New mesage	*Alert user*
Browse	↓	Scroll	*Speak message*

Table 1: Auditory User Interface For IM

Notice that in each of the above instances, one needs to determine the priority of the incoming notification relative to any other ongoing speech activity. Where appropriate, pending notifications may need to be queued for delivery *after* any on-going spoken output has completed.

4. LEVERAGING ARIA LIVE REGIONS

This section illustrates the use of ARIA *live regions* to enable the automatic speaking of IM messages as they arrive. The technique used to produce automatic feedback in the other scenarios enumerated above is achieved in an analogous manner.

During chat interaction, messages arrive in the form of ⟨div⟩ elements that are added to the chat transcript area of the Google Talk interface. Addition of these ⟨div⟩ nodes to the Document Object Model (DOM) result in the automatic updating of the visual canvas. As a result, the visual display shows the newly arrived message.

These ⟨div⟩ elements contain two ⟨span⟩ children. The first ⟨span⟩ element contains the user name; the second ⟨span⟩ element holds the incoming message. Cascading Style Sheet (CSS) properties on these individual ⟨span⟩ elements are used to visually style these portions of the visual display. We use these CSS properties to advantage when correctly segmenting the updated content before audio formatting the result.

As sketched out earlier, we want the incoming message to be spoken as it arrives. This is achieved by tagging the ⟨span⟩ element containing the message body with ARIA property *live=rude*. Presence of this property indicates that the contents of that element should be spoken when changed. A value of *rude* indicates that such spoken feedback should interrupt any on-going speech.

By setting the live property to rude, the AxsJAX enhancement to Google Talk gives adaptive technologies a hint that any update to this message is very high priority; —, it has the highest priority possible. As a result, when this message arrives, the adaptive technology will announce it immediately.

Notice that property *live=rude* is only set on the ⟨span⟩ containing the message body and not the entire ⟨div⟩ element; this is done on purpose in order to restrict automatic reading of the incoming message to only its contents and not its sender. This creates a much more efficient auditory user interface as the user is not forced to listen to redundant information —in a two-way conversation, the user already knows the identity of the chat buddy.

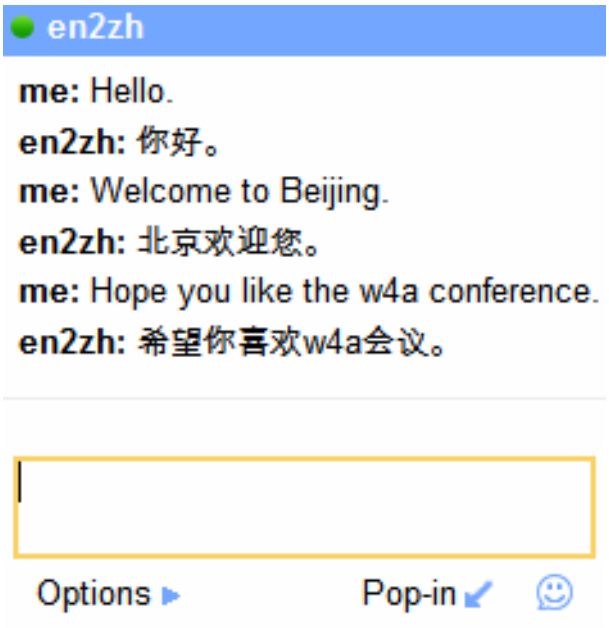

Figure 1: Chat with EN2ZH

5. A TALKING TRANSLATOR

Web-2.0 applications bring the power of hyperlinking to software components deployed on the Web; a direct end-user benefit is the coming together of useful services to create solutions that are bigger than the sum of their constituent parts.

Google offers a Web interface to our machine translation technology; recently, we surfaced this in the form of *chat bots*. As an example, one can converse with a chat buddy of the form `en2zh@bot.talk.google.com` in English and get messages automatically translated to Mandarin. In general, translation bots for the available languages are named using the following template:

xx2yy@bot.talk.google.com

where *xx* is the two-letter language code for the source language, and *yy* is the two-letter language code for the target language.

The integration of Google Talk into GMail brings this service one step closer to the GMail user. With Google Talk enhanced with ARIA support via google-axsjax, chat sessions with the translation-bot produce spoken feedback when used in conjunction with self-voicing browsers or screenreaders that fully implement ARIA.

An immediate consequence is that chat responses from the various translation bots are automatically spoken.

Figure 1 shows a screenshot of a sample interaction with the English to Mandarin translation bot. Here, the user has typed two English utterances as detailed below, each of which has been translated to Mandarin by the translation bot. As the visual display updates with each translated utterance, the user hears the appropriate Chinese utterance.

Me Hello!

Bot Greets me in Mandarin.

Me Welcome to Beijing

Bot Responds with the Chinese translation.

Me Hope you like the W4A conference.

Bot Responds with the Chinese translation.

HTML chat responses from the translation-bot carry an appropriate `lang` attribute that identifies the language of the content. An immediate consequence is that one can type in an English utterance, and listen to the resulting Chinese translation —provided the underlying platform has Chinese text-to-speech installed.

6. CONCLUSION

W3C ARIA is a work in progress and is defining some of the core DOM primitives needed for enhancing the accessibility of Web-2.0 applications. These primitives provide a basic toolbox that can be thought of as the *assembly language* for augmenting Web-2.0 applications. AxsJAX is an Open Source framework that uses these basic primitives to create higher-level design patterns that are better suited for use by Web developers. This framework helps validate and improve the ARIA design, while helping to discover effective design patterns using ARIA. As a useful side-effect, the framework also helps us deliver *immediate* end-user value by enabling us to release early-access solutions that can eventually become a core part of the services being enhanced.

7. REFERENCES

[1] M.D. Center. AJAX: WAI ARIA Live Regions. 2007.

[2] WAI Protocols and Formats Group. ARIA— Access To Rich Internet Applications Overview. Technical report, W3C, 2006. See `http://www.w3.org/WAI/intro/aria`.

[3] TV Raman. Emacspeak —direct speech access. *Proceedings of the second annual ACM conference on Assistive technologies*, pages 32–36, 1996.

[4] TV Raman. *Auditory User Interfaces: Toward the Speaking Computer*. Kluwer Academic Publishers Norwell, MA, USA, 1997.

[5] R. Schwerdtfeger et al. Roadmap for Accessible Rich Internet Applications (WAI-ARIA Roadmap). *World Wide Web Consortium Recommendation Working Draft*, 2006.

[6] P. Thiessen and C. Chen. Ajax live regions: chat as a case example. *Proceedings of the 2007 international cross-disciplinary workshop on Web accessibility (W4A)*, pages 7–14, 2007.