Web Browser Accessibility using Open Source Software

Željko Obrenović
CWI
P.O. Box 94079, 1090 GB
Amsterdam, The Netherlands
zeljko.obrenovic@cwi.nl

Jacco van Ossenbruggen
CWI
P.O. Box 94079, 1090 GB
Amsterdam, The Netherlands
jacco.van.ossenbruggen@cwi.nl

ABSTRACT

A Web browser provides a uniform user interface to different types of information. Making this interface universally accessible and more interactive is a long term goal still far from being achieved. Universally accessible browsers require novel interaction modalities and additional functionalities, for which existing browsers tend to provide only partial solutions. Although functionality for Web accessibility can be found as open source and free software components, their reuse and integration is complex because they were developed in diverse implementation environments, following standards and conventions incompatible with the Web.

To enable the integration of existing partial solutions within a mainstream Web browser environment, we have developed a middleware infrastructure, AMICO:WEB. This enables browser access to a wide variety of open source and free software components. The main contribution of AMICO:WEB is in enabling the syntactic interoperability between Web extension mechanisms and a variety of integration mechanisms used by open source and free software components. It also bridges the semantic differences between the high-level world of Web XML-based APIs and the low-level APIs of the device-oriented world.

We discuss the design decisions made during the development of AMICO:WEB in the context of Web accessibility, using two typical usage scenarios: one describing a disabled user using a mainstream Web browser with additional interaction modalities; another describing a non-disabled user browsing in a suboptimal interaction situation.

Categories and Subject Descriptors

K.4.2 [**Computers and Society**]: Assistive technologies for persons with disabilities; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces; D.2 [**Software**]: Software Engineering; K.4.2 Computers and Society [**Social Issues**]: Handicapped persons / special needs

W4A2007 - Technical Paper, May 07–08, 2007, Banff, Canada. Co-Located with the 16th International World Wide Web Conference.

General Terms

Human Factors, Design, Standardization

Keywords

Web Accessibility, User Interfaces, Open Source Software, Software Platform, Middleware

1. INTRODUCTION

Web 2.0 applications typically have users browsing *and* creating content. Accessible Web 2.0 software has thus not only to support the traditional output modalities, but also a rich variety of input modalities that go beyond the typical link and form based interaction of the first generation Web tools. Although Asynchronous JavaScript and XML (AJAX) and similar GUI approaches labelled as Web 2.0 may provide richer interfaces for the average user, its use often results in poorer interfaces in terms of accessibility. We feel that the same technology underlying current AJAX interfaces in combination with novel interaction modalities could also be used to improve accessibility, without harming the AJAX-based rich interface for non-disabled users.

The main problem is that AJAX and similar approaches currently do not have access to interaction modalites beyond keyboard and mouse interaction supported by browsers. The open source community has, on the other hand, developed a number of freely available components that can be used to enrich Web interaction. Examples include libraries for vision-based interaction modalities [9, 20], lexical tools [5, 30], and speech input and output for many languages [8, 14, 17, 24]. A natural question arrives: can we borrow from these solutions, and reuse them in combination with Web (2.0) technologies?

Open source software (OSS) components are, however, developed for other purposes, in diverse implementation environments, following standards and conventions often incompatible with the Web. As a consequence, using open source and free software components often comes with a high price [11]: even though you get OSS components for free, you usually have to invest significant time and effort to integrate them, as your team needs to master at least several programming languages, integration interfaces and network protocols. This is not acceptable for many Web developers, who cannot be expected to work under unrealistic economic models in which they are required to "spend too much and receive too little", supporting the needs of a small number of users [22]. The biggest price, however, is paid by the users who are still unable to access the Web efficiently.

To address these problems we developed AMICO:WEB, an infrastructure that facilitates efficient reuse and integration of OSS components into the Web environment. The main contribution of AMICO:WEB is in enabling the syntactic and semantic interoperability between Web extension mechanisms and a variety of integration mechanisms used by open source and free software components. Its design is based on our experiences in solving practical problems where we have used open source components to improve accessibility of rich media Web applications.

The remainder of this paper is structured as follows. We first introduce two sample scenarios that are used to illustrate the problem and potential solutions. We discuss the pros and cons of existing solutions and discuss other related work. We give a high level overview of the AMICO:WEB infrastructure and describe how this infrastructure is used to implement solutions to the problems of the two sample scenarios. Finally, we discuss some of the open issues, and conclude with plans for future work.

2. EXAMPLE SCENARIOS

We describe two accessibility scenarios that we use to illustrate the problem and proposed solution. The first scenario describes Michelle, a disabled user using a standard Web browser with additional interaction modalities. The second describes Simon, a non-disabled user using a browser that helps him to use foreign language Web sites. Note that both use cases deal with a range of very specific problems, but are intended to represent a wide variety of accessibility issues. They are also intended to show that many accessibility issues are not specific to disabled users. For example, Web sites that are accessible for users with bad eyesight are often also accessible for users with good eyesight that need to interact with the same site in poor lighting conditions.

2.1 Scenario 1: Michelle

Michelle has a spinal cord injury, and she cannot use her hands sufficiently well to control a keyboard or mouse. She uses a mainstream Web browser that has been enhanced with camera-based input and a speech-recogniser to help her access the Internet. A camera-based face detector tracks the position of her face and her proximity to the screen, while a camera-based motion detector tracks the direction and intensity of her head movements. Using these modalities, Michelle can, for example, scroll the page by moving her head up or down, or control the playback of Youtube movies with movements. Both modalities use her ordinary Web camera. The speech recogniser enables her to navigate Web pages, follow links, and give common Web browser commands. Although she uses the browser most of the time without the keyboard or mouse, when she is using the keyboard, the browser checks her input, and suggests spelling corrections.

The browser also offers Michelle some (limited) interaction with 3D and multimedia content on the Web. For example, she can examine an X3D/VRML scene by moving hear head left or right, and walk through the scene by moving her head forwards or backwards. She can also go to a specific 3D viewpoint using speech. In the case of multimedia content, the browser detects embedded movies or music clips in the page, enabling her to use speech to start, stop or pause the playback, and to change the volume. When she temporarily leaves her position, the browser face detector recognises that there is no one in front of the screen, so it pauses the playback, resuming it when she returns. This is better than using speech, which works less reliably because of the background sound from the multimedia content.

2.2 Scenario 2: Simon

Simon is a French student who recently moved to the Netherlands. He has a solid knowledge of English and has started to learn Dutch. His knowledge of Dutch, however, is still not sufficient to enable him to use Dutch Web sites efficiently. If he wants to use e-banking services, for example, he can do so reliably only if he uses Dutch version of his bank Web site. Also, he wants to improve his social life by joining sports or dancing clubs, which, as they have mostly Dutch members, offer only Dutch texts, with optional out-dated and less detailed English versions.

Luckily, his new browser extensions offer him help. While interacting with a Web page, he can select the text and hear the translation of it from a text-to-speech (TTS) engine in English or French. This is much better than using online or offline dictionaries as they require him to lose the focus of his work. Simon also prefers this to usage of online translation services, which can translate whole pages on-the-fly, but which almost always change the layout of the page, or remove part of its functionality, especially if the Web site uses scripts or complex session management. Also, as his Dutch improves, he wants the system to help him with translations less and less.

The browser remembers his translation requests, saving phrases he asked for, the URI of the page, date and time. When he next visits the page, the browser shows him the phrases that he has requested in previous visits. He can also use this as a learning tool, getting a list of most frequently asked phrases, or, for example, the list of words from the last week asked more than twice.

In addition, the browser offers him speech input services. For example, when filling in online forms he can say the word in English, while the browser translates it into Dutch and enters the translated word in the field.

3. EXISTING SOLUTIONS

Both of the presented scenarios require novel interaction modalities and browser functions. Some of the existing solutions can be used to implement a part of these scenarios. In this section we identify these solutions, and discuss their main limitations.

The most visible outcome of the W3C Web Accessibility Initiative (WAI) [26] is a set of standards, guidelines, and checklists, provided within. These documents provide a common understanding of the Web accessibility issues, helping designers and developers to meet proposed requirements. W3C also runs several activities directly or indirectly related to Web accessibility, to support interaction modalities beyond the keyboard and mouse. For example, the MultiModal Interaction (MMI) activity [27] covering input via speech, handwriting and keystrokes, and output via displays, pre-recorded and synthetic speech, audio, and tactile mechanisms, such as mobile phone vibrators and Braille strips. The W3C Speech Interface Framework [29] further extends speech interaction, covering issues of voice dialogs, speech synthesis, speech recognition, telephony call control for voice browsers and other requirements for interactive voice response applications, including use by people with

hearing or speaking impairments. The W3C Device Independence initiative promotes the idea of "Access to a Unified Web from Any Device in Any Context by Anyone" [25]. However, experience shows that these guidelines have not been successful in producing accessible Web sites [16]. W3C initiatives on alternative interaction modalities cover just a subset of modalities required by accessibility solutions, such as speech and handwriting. Moreover, it is still not clear how and when they will be implemented. One of the problems is a requirement that components have to be built according to specified interfaces, which can be a high threshold for many developers.

Besides these standardization and regulation efforts, there are also solutions related to our work that attempt to make a more accessible Web by adapting and extending existing (non-accessible) Web components. We classify these approaches into four groups: general purpose accessibility tools, browsers specifically designed for disabled people, adapted open source Web browsers, and browser extensions and plugins.

3.1 General Purpose Accessibility Tools

General purpose accessibility tools, such as screen readers, can be used with any application, and consequently with Web browsers. New versions of operating systems, such as Windows XP, include speech input modality and accessibility tools such as a narrator or magnifier. In early work on Web accessibility, Asakawa et al. used ScreenReader/2, a screen reader for OS/2, with Netscape Navigator [1].

The problem with general purpose tools is that they can access the application only through relatively low-level application interfaces provided by the operating system. Therefore, these tools cannot easily understand the structure of the presentation, nor access all the functions that the application provides. For example, Simon can use these tools to "talk" with the Web browser, but speech interaction with generic speech commanders is usually limited to menu items, and it is not adaptable to the content of the Web page.

3.2 Specially Designed Browsers

Several browsers specifically have been designed for people with disabilities, such as BrookesTalk [31] or pwWebSpeak[1] (see [28] for an overview).

Specially designed browsers have not been very popular solutions, as users normally prefer to use standard browsers with accessibility adaptations, rather than specially designed browsers, which also label them as "disabled" [22]. Specialised browsers also tend to be less rich in functionality, and cover just a small part of the functionality necessary for the implementation of our example scenarios.

3.3 Adapted Open Source Browsers

The availability of source code for browsers such as Mozilla[2] has resulted in several approaches that have created adapted browser versions with added accessibility functions. For example, as part of the *accessibilityWorks* project, Hanson et al. [10] provided software enhancements to the Mozilla browser to allow users to control the accessibility features of their browsing environment. They developed user controls to support a number of adaptations that can increase the usability of Web pages for a diverse population of users, including transformations that can change page presentation, mouse and keyboard input correction, as well as vision-based control for users unable to use their hands for computer input. Their work focuses on a limited set of user controls that facilitate adaptations that can increase the usability of Web pages for a larger population of users. A subset of the functions of this browser, such as vision-based control, can be useful for our sample scenario with Michele.

The main problem with adaptations based on changing the source code of Web browsers is the complexity of development and maintenance of such code. Changing browser code requires significant developer effort, and a solid knowledge of implementation details of the Web browser. This kind of adaptation is usually feasible for companies that can invest significant resource in such efforts. Maintaining multiplatform support and installation is an additional concern, while reuse of results in a new version of the browser can require a significant effort. Reuse and integration of other components is limited by the implementation platform that the browser uses.

3.4 Browser Extensions

Browser extensions, such as the Mozilla / Firefox Accessibility Extension[3], are aimed at making it easier for people with a disability to view and navigate Web content. It can also be used by developers to check their structural markup to make sure it matches the page content. Another popular extension is the Greasemonkey Mozilla/Firefox extension[4] that allows users to apply javascript and DHTML "user scripts" to any Web page. Greasemonkey enables users to add fragments of DHTML ("user scripts") to any Web page to change its behaviour. User scripts can control any aspect of a Web page's design or interaction through the Firefox API. Greasemonkey scripts can be applied to all pages, or just selected ones, based on the URI pattern provided with the scripts. It has been used to improve accessibility Web page behaviour to display access keys, add shortcut keys to common page function (such as previous and next links or navigation), remove pop-up windows, and allow image resizing and page zooming. Fire Vox is another open source Firefox/Mozilla extension[5], which works as a screen reader designed especially for Firefox. It supports features, such as identifying headings, links and images, providing navigational assistance, and support for MathML and CSS speech module properties. It currently uses the FreeTTS [8] engine output.

Toolbar extensions, however, cannot solve the problem of providing alternative input modalities beyond browser supported keyboard and mouse input, needed for both of our example scenarios.

4. WEB ACCESSIBILITY: POTENTIAL OF OPEN SOURCE AND FREE SOFTWARE

Existing solutions cannot be used to fully implement the scenarios presented in Section 2. For these scenarios it is necessary to use many components and services, both locally and remotely. Only some of these components are available within current Web browsers. On the other hand, there are

[1] `http://www.soundlinks.com/pwgen.htm`
[2] `http://www.mozilla.org/`
[3] `https://addons.mozilla.org/firefox/1891/`
[4] `http://greasemonkey.mozdev.org/`
[5] `http://www.firevox.clcworld.net/`

thousands of open source projects that cover issues useful for Web accessibility, and that provide necessary functionality to implement our example scenarios. For example, the open source community has developed a huge number of components that can be used to support novel interaction modalities. Just for the domain of computer vision, there are hundreds of freely available computer vision software modules[6]. However, reusing these components within the Web environment is not an easy task.

In this section we identify the main challenges in reusing OSS component. We also identify the basic requirements for integration middleware infrastructure that can simplify reuse of OSS and free components and services in the Web environment.

4.1 Reuse and Integration of OSS and Free Software

Reuse and integration of OSS and free components within the existing Web infrastructure is complicated because they are developed for other purposes, in diverse implementation environments, and therefore cannot be easily embedded into the Web components. OSS projects mostly cover the needs of their developers, while the addition of features and other modifications is driven by the interest and wishes of the contributors [13]. The choice of development environment and supported platforms is also very diverse and driven by the needs and experience of contributors. The main question, therefore, is how to combine components that use a wide variety of (often low level) protocols and application programming interfaces with incompatible (high level) standards, such as those used on the Web.

The basic idea of our approach is the development of an infrastructure that can exploit the potential of available OSS components and provide a flexible mapping between existing Web extension mechanisms and OSS integration mechanisms. The introduction of a flexible middleware infrastructure, to facilitate integration of components, brings many advantages to developers as it can significantly speed up application development [12].

Using some of the existing component-off-the-shelf (COTS) middleware systems is practically impossible for most open source components, as they usually use very diverse technologies, protocols, and implementation platforms, which makes it hard to wrap the code in COTS compliant packages. Existing systems for loosely integration of components [7] are aimed at collaborative applications or context-aware computing, and cannot easily be used in other domains. Furthermore, they support only a limited number of integration interfaces.

Our approach is similar to that taken by Miller et al. [15], where they extended their LAPIS[7] experimental Web browser with an interface to the UNIX command shell, in order to help users automate HTML interfaces creating pipelines of Web services and local programs. With our approach, however, we want to reuse more services, supported on different platforms, and to extend standard and widely used browsers.

4.2 Requirements for Integration Infrastructure

[6] `http://www.cs.cmu.edu/~cil/v-source.html`
[7] `http://groups.csail.mit.edu/graphics/lapis/`

The basic requirement for our infrastructure is an extension of standard main stream Web browsers through *existing* extension mechanisms. Users prefer standard browsers with supplementary accessibility adaptations, rather than browsers specially designed for users with disabilities [22]. As we have already mentioned in Section 3.2, specialised browsers provide only a limited set of functions, and tend to "label" users as being disabled. Secondly, existing Web extension mechanisms, such as browser toolbar extensions, scripting and plugins, provide powerful and sophisticated interfaces to browser functionality, while the development of extensions is usually much easier than approaches, such as, changing the source code of the browser. Moreover, extensions can be installed by an ordinary user in a few steps.

In addition to these basic requirements, and in order to support wider reuse of available software and services, based on our experiences with reusing OSS, we have identified the following requirements that such an infrastructure has to fulfil:

- *Support multiple integration interfaces, and enable integration of new interfaces.* Components from open source projects use diverse integration interfaces, such as XML-RPC, OpenSound Control, HTTP, TCP, of which none is predominant. Adapting components to one common interface is not an easy task, and sometimes not possible, as OSS components are developed in diverse implementation environments.

- *Bridge semantic and temporal gaps.* Low-level components, such as sensors, and higher level components, such as Web services, work with significantly different data structures and temporal constraints. For example, sensors, such as a face detector, can send dozens of UDP packages per second with simple data structures about detected events. Web services, on the other hand, use more complex HTTP protocol and complex XML encoded data, with delay which is sometimes measured in seconds. To enable integration of components that work with significantly different data structures and temporal constraints, the infrastructure has to be able to abstract and map different data types and to support temporal functions, such as frequency filtering.

- *Enable flexible and reusable integration.* One of the weaknesses of existing component integration systems is that, when using them, developers usually have to agree on many rules, such as naming conventions, if they want to connect components. The infrastructure should not require such agreements and should provide the definition of the connection among components that can bridge this semantic gap and which can (at least in part) be reused.

- *Dynamic integration and fault tolerance.* To support various user needs, the platform has to support dynamic fault-tolerant reconnection of modules, possibly depending on the current state of the system. Optimal interaction configuration can also differ from user to user. Fault tolerance is necessary as many of the components are still under development and are often not very stable.

- *Use open standards.* OSS developers have a body of standards that multiple vendors have agreed upon.

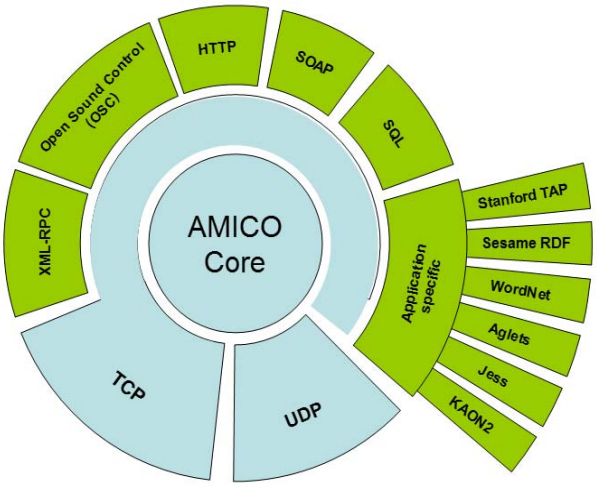

Figure 1: AMICO:WEB and application interfaces.

> Standardisation provides a significant driving force for further progress because it codifies best practises, enables and encourages reuse, and facilitates interworking between complementary tools.

5. AMICO:WEB

The main contribution of our work is the development of a brokering infrastructure that connects Web extension mechanism with a wide variety of interfaces found in many OSS components. The AMICO:WEB infrastructure enables developers to rapidly prototype and experiment with various OSS components in a Web environment. It consists of two main parts: a brokering infrastructure and Web integration mechanisms. The brokering infrastructure, called AMICO (Adaptable Multi-Interface COmmunicator), is a generic platform, used to support rapid prototyping with OSS components in different domains [19]. It was designed to meet the above requirements for reuse of heterogeneous OSS components AMICO:WEB is the extension of this platform into the Web domain. The infrastructure and some examples are available at the SourceForge Web site [8]. In this section we describe the design of the AMICO:WEB brokering infrastructure and its Web integration interfaces.

5.1 The Brokering Infrastructure

The proposed brokering infrastructure is based on the publish-subscribe design pattern. It is well suited for integration of loosely-coupled parties, and often used in context-aware and collaborative computing. A publisher may update a shared data repository without being concerned with whether any subscribers are listening for updates. When using simple data structures, the loosely coupled approach can be highly adaptable, so that new applications can both reuse existing data in the repository and add their own data without breaking the infrastructure. This approach is also fault tolerant, as components run as independent processes. In the loosely coupled model, components can run on different machines in a distributed environment. Components communicate by exchanging events through a shared data repository consisting of named slots. Components can update the slots, and register for notifications about changes.

A key difference between our infrastructure and regular notification services is based on our requirement for supporting more than one integration interface. AMICO:WEB provides a unified view on different communication interfaces, based on a common space to interconnect them. As Figure 1 shows, we support several widely used standard communication protocols. AMICO:WEB is extensible, and it is possible to add new communication interfaces. Most of the communication adapters are bidirectional. For example, an XML-RPC communication interface may run an XML-RPC server, enabling other modules to update and read data through this interface, and it also enables the definition of XML-RPC adapters that map this data to parameters of method calls on other XML-RPC servers. It is also possible to directly communicate with AMICO:WEB using a TCP connection, or by sending UDP packages.

The infrastructure enables a flexible configuration of the integration through instantiation of new, derived slots from slots directly updated by the components. Therefore, applications do not have to directly use slots updated by other modules, but they can add transformations that adapt these values for their own needs. New slots are derived by using a set of transformations defined in XSLT [4], a widely used and rich transformation language supported by many existing tools. Transformations can also be reused in different scenarios. The transformations provide the means for bridging the different levels of abstraction found in the different data structures used by each module. For example, they can abstract low level data so that we, instead of receiving all low level events, receive notification only when, for example, user enters some area in the camera visual field, or when detected movement activity is above or below given thresholds. They can also solve the problem of different temporal constraints of components. For example, transformations can use time stamps to derive new slots that are updated with lower frequency, allowing higher level modules to ignore other updates. In this way modules can be simpler as they do not have to be changed to meet the synthetic and temporal constraints of other modules, and can be reused in other situations.

AMICO is realised as a Java application, and has been tested on several operating systems.

To connect our brokering infrastructure with the Web on client, proxy and server side, we have developed several integration components. The focus of work in this paper is on integration with the browser side. In Section 7.3 we briefly discuss integration mechanisms for proxy and server components.

5.2 Browser Integration Mechanisms

We use several browser extension mechanisms in combination with AMICO:WEB: Firefox/Mozilla extensions, complex scripting libraries such as AJAX, and applets.

5.2.1 Firefox/Mozilla Extensions

Based on the MIT Simile open source Java Firefox Extension[9], we developed a generic AMICO:WEB Firefox/Mozilla extension. The Firefox extension mechanism gives us access to browser functionality and through it we can also read and change the content of a Web page dynamically. Within

[8] `http://amico.sourceforge.net`

[9] `http://simile.mit.edu/java-firefox-extension/`

```
<script>
    function p_start() { document.movieclip.DoPlay()      }
    function p_pause() { document.movieclip.DoPause() }
    function p_stop() { document.movieclip.DoStop()      }
    function p_set_volume(volume) {
            document.movieclip.SetVolume(volume)
    }
    function setColor(color) {
            document.bgColor=color
    }
</script>
.....
<applet code="ScriptingApplet.class"
                    width="0" width="0" MAYSCRIPT>
  <param name="port" value="3320">
  <param name="command-template-1"
           value="ADD TEMPLATE DIFF action[.='play'] p_start">
  <param name="command-template-2"
           value="ADD TEMPLATE DIFF action[.='pause'] p_pause">
....
  <param name="command-template-N"
          value="ADD TEMPLATE DIFF color setColor <%=color%>">
</applet>
```

Figure 2: Parameters for the *ScriptingApplet* used to control the playback of the RealPlayer plugin embedded in an HTML page.

the toolbar extension, we have added a thread that opens a TCP connection to AMICO to send and receive events from the infrastructure. The communication is bidirectional, e.g. the toolbar maps the events from the infrastructure to the calls of the browser API, and it can update AMICO:WEB with data from the browser. In Section 6 we describe how we have used this mechanism to enable browsers to communicate with online translation services, databases and TTS engines.

Other browsers, such as Internet Explorer, also allow developments of customised extensions. We have focused, however, on extensions for Firefox/Mozilla due to its multiplatform support and because it is available as an open source project.

5.2.2 Complex Scripting Libraries

Scripting libraries used within HTML pages offer more and more interactivity and connectivity. For example, AJAX can access the server side without reloading the page, and scripting functions can also use this mechanism to access AMICO. To do so, AJAX functions can use XMLHttpRequest object, and access AMICO through the AMICO HTTP interface. AMICO returns an XML encoded list of requested values. The limitation of this approach is that it can only be used in pull mode, i.e. scripts can update or request any values from AMICO, while they cannot receive notifications from AMICO.

5.2.3 Applets and Scripts

To enable fully bidirectional communication between AMICO and the browser, we have worked on other approaches, such as combined usage of Java applets and scripting. We have developed two applets for this purpose: *ContentChangerApplet* and *ScriptingApplet*. Both applets open a TCP connection to AMICO. *ContentChangerApplet* registers for any variable that contains a Web link and it reloads the content of the Web browser when the variable is changed. The link, for example, could be provided by a user who speaks the phrase associated with an often visited Web site. *ScriptingApplet* specifies the mapping between AMICO notifications and calls of HTML scripting functions. *ScriptingApplet*, therefore, transforms notifications sent from the communicator into calls of script functions, and in the same time enables script functions to update AMICO:WEB. The applet is completely reconfigurable through PARAM tags, and developers can use it to control any scriptable element in the HTML page. It is also very small (about 6kB compiled), so its usage within the page does not introduce significant delay. *ScriptingApplet* can be used with any scripts including complex scripting libraries such as AJAX. We have used our applet to control interaction in Mozilla, Firefox, and Internet Explorer Web browsers, including:

- Controlling standard elements of HTML page through DOM interfaces; changing content, such as text in a paragraph; changing style elements, such as background colour; or changing the layout, such as moving image controls by a camera.
- Controlling playback in multimedia movie and music plugins, such as RealPlayer, Windows Media Player, and QuickTime players.
- Accessing and manipulating VRML/X3D scenes, using Scene Authoring Interface (SAI) scripting API exposed by VRML/X3D players [6].

Figure 2 illustrates parameters of the scripting applet used to control the playback of the RealPlayer plugin. The applet registers for changes of data slot named "action" (which can have values "start", "pause", "stop") and variable "color". Whenever these data change, the applet receives a notification and starts the appropriate script function.

6. USING AMICO:WEB TO IMPLEMENT THE EXAMPLE SCENARIOS

We have used AMICO:WEB to implement the two example scenarios described in Section 2. With these two scenarios we illustrate how relatively complex and novel use cases can be implemented using AMICO:WEB along with existing open source and free software components without radical changes in the Web interface design. Both scenarios support client side integration, and currently work only on Firefox / Mozilla browsers. All components used in these scenarios are available for download from our SourceForge Web site, as well as video material demonstrating the use of these components[10].

6.1 Used OSS and Free Components

For processing of human input by camera we have used face and motion detectors based on the OpenCV computing vision library [20], and the HandVu gesture recogniser [9]. The face detector detects number of faces, their size and position. The motion detector detects intensity and direction of motion. HandVu recognises several hand gestures, and it can track the position and angle of the hand.

For speech input and output, we used a English speech recogniser based on Sphinx-4 [24], and three open source TTS engines: the FreeTTS English TTS engine [8], the NEXTENS Dutch TTS engine [17], and the Mary TTS engine that currently supports English, German and Tibetan [14].

[10] `http://amico.sourceforge.net/amico-demos.html`

For storage and querying of data, we have used our SQL interface to a relational user database in MySQL, and our Sesame RDF interface to access a Sesame RDF triple store [23].

We have also used two remote Web services: Yahoo BabelFish translation service[11], and the Google spelling checker service[12].

The listed components and services use very different technologies. We had to combine several protocols and integration approaches to be able to deploy them seamlessly in a single browser session. All components run as independent services, connected with our infrastructure through some of the supported interfaces. For the face and motion detectors we have implemented C++ adapters using a socket library, where we have modified the original OpenCV programs to send detected events as UDP packages to AMICO. We access the HandVu system through the HandVu TCP protocol. The Mary and Nextens TTS engines also use a TCP-based interface, with their own application specific protocols. For the FreeTTS engine, AMICO has been extended with a Java wrapper, communication between AMICO and the wrapper is also over TCP. ConceptNet uses an XML-RPC interface, and the Google spelling checker uses a SOAP interface. For access to SQL databases we have used existing ODBC and JDBC drivers, and for access to the Sesame RDF repository, we have used Sesame's Java API. The components are relatively simple, and unaware of other components. This context independence enables their reuse in other scenarios.

6.2 Implementation of Interaction Modalities for Example Scenarios

The components and services described in the previous section have been arranged in different configurations, and connected to the browser to support the interaction modalities needed for the example scenarios.

Figure 3 shows the basic configuration for the implementation of the interaction modalities in Michelle's scenario. The face detector, motion sensor and gesture recogniser (see Figure 4) are used exclusively, that is, only one of these modules is loaded at any time, depending on the desired interaction. Modules can be loaded and unload during an interaction session, or this can be configured before the modules are started. To support this flexible component loading, we use *ProcessRunner*, an auxiliary tool that is a standard part of the AMICO platform. A Sesame RDF database contains additional data about user profiles, such as user language and other preferences.

This basic configuration is connected to the browser in several ways. The AMICO Firefox extension directly accesses the browser functionality, and maps commands derived from input modalities into actions such as scrolling the Web page, or copying text. Data captured by sensors is also communicated to the *ScriptingApplet*, that is embedded in Web pages by Greasemonkey at the client side. We have developed several Greasemonkey user scripts that enable this basic configuration to be used with various Web sites.

With the Google spelling checker, we enabled a user to, after typing the content, asks for correction of the text. This modality is again integrated in a browser using the *ScriptingApplet*. Using Greasemonkey, we search the page for all textual fields, and add scripts that capture the *onfocus* events, in order to determine the field currently in use. When the user requests a correction, we read the content of the active field, send it to AMICO:WEB, which than contacts the Google spelling checker through Google's SOAP interface. AMICO:WEB parses the XML response from the Google service, and sends the corrected text back to the *ScriptingApplet* which updates the field with the spelling suggestion.

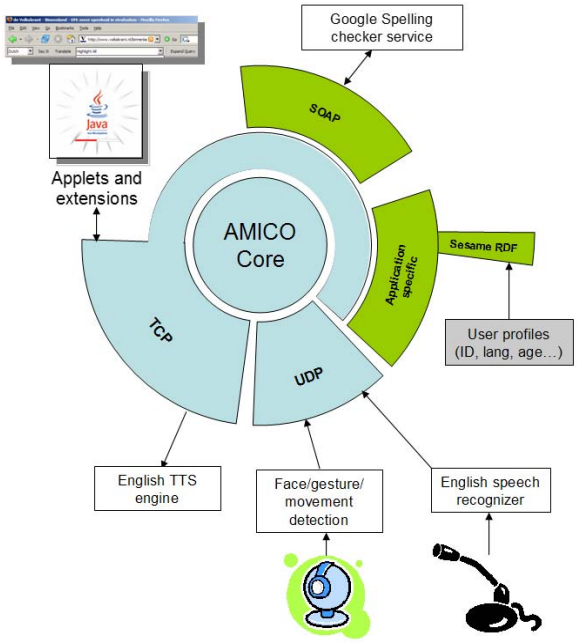

Figure 3: AMICO:WEB configuration of Scenario 1.

Figure 4: Screenshots of three visual input modality components: face detector, motion detector, and gesture recogniser.

Figure 5 shows the basic configuration of Simon's scenario. The external components are integrated with the Web browser through a toolbar extension (Figure 6), that was developed for this scenario by extending the standard AMICO Firefox Extension extension described in Section 5.2.1.

In this interaction scenario, Simon can browse any Web site, select a word, phrase or whole sentence, and use his browser's toolbar to request translation or pronunciation of the text. When he requests translation, the toolbar updates the corresponding variable inside AMICO:WEB. AMICO:WEB then contacts the BabelFish translation service through its HTTP interface, requesting the translation of the selected text, from the language selected in the toolbar, to Simon's preferred language which is defined in his AMICO:WEB user profile. AMICO:WEB receives the HTML

[11] `http://babelfish.yahoo.com/`
[12] `http://code.google.com/apis/soapsearch/`

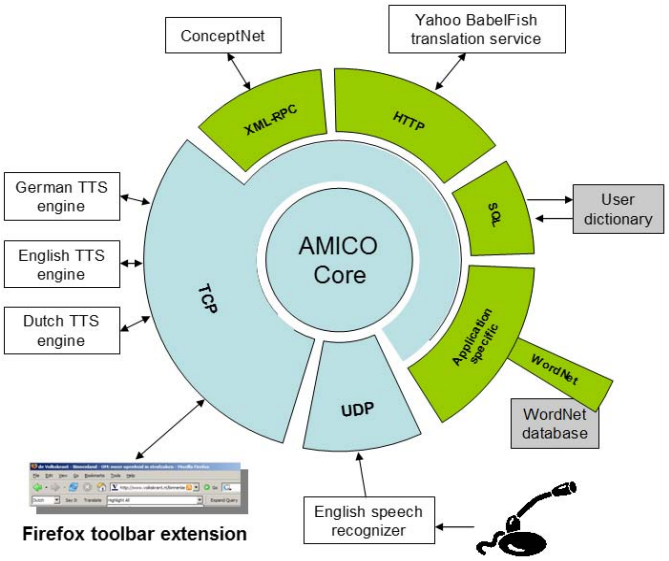

Figure 5: AMICO:WEB configuration of Scenario 1.

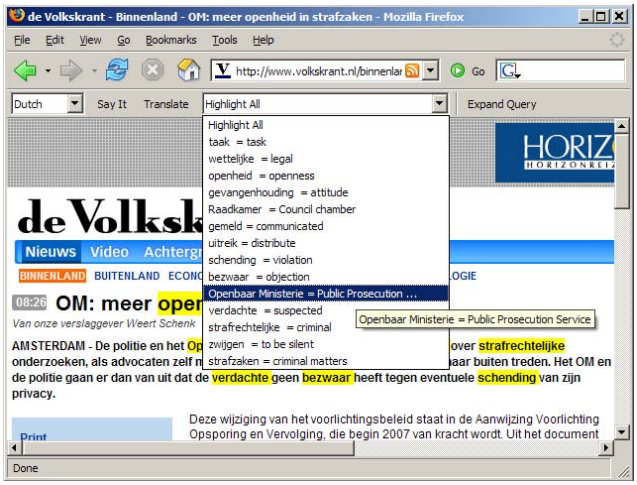

Figure 6: AMICO's Firefox toolbar extension supporting Scenario 2.

encoded response from the BabelFish service, which is then translated by AMICO:WEB's XML processor into valid XHTML code (based on JTidy library[13]). The XML processor then extracts translated text based on given XPath expression, and translated text is sent to the appropriate TTS engine that pronounces the translated text. The full functionality of the system is currently limited by the languages supported by the BabelFish translation service and by the availability of the TTS engines.

Each request for translation is recorded in the SQL database, with data about original and translated text, page location, and date and time of translation. When the user again visits the same page, previously requested translations are presented in the combo box of the toolbar (as shown in Figure 6). Words selected in the combo box are highlighted in the text. In preliminary evaluations, the average time for translation was about 1s, mostly caused by the delay of calling out to BabelFish.

With speech input, translation is supported in a similar fashion. A speech recogniser updates an AMICO:WEB variable with the recognised text. AMICO:WEB than sends the text to the BabelFish translation service, extracts the translated text, and than sends translated text to the toolbar extension that inserts it in a currently active text field in HTML form. Optionally, translated text could also be pronounced by a TTS engine.

7. DISCUSSION

Our approach extends and adapts standard Web components and can be used on a wide variety of Web sites as it does not require specialised markup of the Web pages. However, there are still many open problems of which we discuss three: hiding the complexity of infrastructure from the user, practical problems with applets and scripting, and proxy and server integration mechanisms. We discuss these issues based on early practical experiences in usage of AMICO:WEB in several projects and teaching.

[13] `http://jtidy.sourceforge.net/`

7.1 Hiding the Complexity of the Infrastructure from the User

Using the OSS components in the proposed manner comes with a price, which is mostly connected to the ease of installing and adjusting the adaptation to the user. Although developers could provide AMICO:WEB solutions as well packaged solutions and unified interfaces to underlying components, it is also necessary to install software that cannot be packaged with AMICO:WEB. In some cases we were able to package OSS components with our infrastructure and a single install procedure. In other cases, however, it is necessary that the user downloads and configures additional third party components. This can be a serious problem if the installation procedure of these components is not straightforward.

Second problem is how to hide the complexity of running diverse OSS components from the users. In our approach, most of the OSS components run as independent services, connected with our platform through one of the supported interfaces. While AMICO:WEB provide solutions connecting these components, they still all have to be started and come up before the system works. Asking a user to manually start all the necessary components and services is definitely not an option. Therefore, as part of our platform, we provide several auxiliary tools, that hide the complexity of the infrastructure from the user. One of such tools is a flexible OSS component loader, that dynamically loads the necessary components based on a given configuration script. End-users see only resulting behaviour of the system, while the tools run the necessary software in the background.

An additional problem is component reconfiguration during an interactive session. The simplest approach to enable dynamic reconfiguration among components is to start all the necessary components at the same time and then just reconfigure them when necessary. Some components, however, require exclusive access to resources such as a camera. In other cases approach introduced unacceptable performance and memory overhead. To solve this problem, we have enabled the infrastructure to dynamically start and

stop necessary components, based on, for example, selection of modalities from the browser toolbar.

Additionally, we have to solve the problem of giving the user feedback via the web browser about the current state of the platform. In our tests, starting some of the OSS components, such as TTS engines or speech recognisers, sometimes took 10 seconds or more, so it is important that the user knows when s/he can actually start to use the added functionality. One approach that we have applied is to use some elements of the browser interface, such as the toolbar extensions, to notify the user about the state, but this is only possible when the user is using the AMICO:WEB toolbar extensions. The other is to use background communication channels, such as speech output, to inform the user, but that requires running a TTS engine, so it is appropriate only when the TTS engine is part of the configuration and already running. There are still open problems in this area, and in future work we will work more on making the infrastructure and its installation and configuration more transparent and accessible.

7.2 Solving Practical Problems with Applets and Scripting

There are two important problems that we have had to solve in order to make usage of our applet integration interface practical. Firstly, AMICO:WEB usually runs on the client machine, as it has to connect to input devices, such as cameras or microphone, and the *ScriptingApplet* has to be able to connect to them. However, due to security limitations, applets, if not signed, can communicate only with the applications on the machine where the applet codebase is [18]. The codebase does not have to be the same as the Web site of the page. Therefore, to simplify usage of our applet, and to avoid the need for running the Web server at the local machine, we have embedded a simplified Web server in the AMICO:WEB HTTP interface, which returns the applet code on browser requests. This approach is also more secure then signing the applets, as our applet now can communicate only with local machine. The AMICO:WEB, on the other hand, can be run in the Java sandbox configured to reject applet connections outside the local machine.

A second problem is that we must somehow embed applets in the Web page. For some of our own Web sites, we did this manually, but this approach cannot be applied to other sites. Applets can also be embedded on-the-fly using proxy handlers or toolbar extensions. However, this approach is not practical due to the complexity of parsing of HTML code in the proxy server handlers, and it requires user to run a personalised proxy on their machine. This approach is also limited to generic functions, as the semantics of the Web page are hard to understand from the HTML text. The most flexible and practical way, although limited to Firefox/Mozilla, proved to be the usage of our *ScriptingApplet* in combination with the Greasemonkey toolbar extension, described in Section 3.4. Even when it is possible to manually embed the applet in the HTML page, this approach allows more flexibility and is supports personalisation, as each user can apply the adaptations that suits them best.

7.3 Proxy and Server Integration Mechanisms

AMICO:WEB can also be used with proxy and server-side components. We have connected our platform with several open source Java proxy servers, such as Paw (pro-active Web filter) [21] and the Muffin WWW filtering system [3], with proactive filtering, which allow plug-in integration of custom filters and handlers. These filters and handlers can monitor and even modify all the data requests and response parameters. Proxies and AMICO:WEB can work together in several modes. A proxy handler can analyse a Web page, and update the AMICO:WEB with data about the page, which can than be used by other modules connected to AMICO:WEB. On the other hand, proxy handlers and filters can register for notification about parameters from AMICO:WEB, and change their content accordingly. Proxies can also serve as a means of embedding *ScriptingApplet* to enhance client-side interaction in existing content.

Active server pages can work with AMICO:WEB in a similar bidirectional way. Server pages can query AMICO:WEB for parameters that can be used to generate the content, for example, to adapt the size of pages according to user preferences. Web servers can also send information about a Web page that is not present in the metadata within the page. For example, the server may send AMICO UDP packages containing metadata about social cues from Web log files about the current Web page, such as total number of visits, average per day, the number of visitors that are currently looking at the page.

8. CONCLUSION

We have presented our experiences in reusing open source and free software to add additional accessibility functions to existing Web components. We have developed an infrastructure that facilitates efficient reuse and integration of OSS components in a Web environment. Our approach focuses on reuse and rapid prototyping, and therefore complements existing approaches, such as changing the source code of the browser, which require significantly more developers' resources. Our approach is not aimed at replacing existing approaches, but at providing a solution in situations where it is not effective to use more homogeneous approaches, due to, for example, the high price of development.

By connecting the infrastructure with existing extension Web mechanisms, we were able to support many of the accessibility scenarios on the Web by integrating existing solutions. We illustrate the proposed approach on two example scenarios, one describing a disabled user using a standard Web browser with additional interaction modalities, and the other describing a non-disabled user using a browser in a suboptimal interaction situation. Our approach is also suitable for rapid prototyping of various other accessibility improvements, which can then be evaluated early in the development cycle.

We have started a collaboration with the School of Interactive Arts and Technology at Simon Fraser University in Surrey, Canada, to use our approach to improve accessibility of Web based e-learning systems. At the Free University in Amsterdam, we have started to use this infrastructure to teach students how to rapidly develop solutions with novel interaction modalities, especially with rich multimedia content and X3D/VRML interaction environments. We are working with the V2_ Institute from Rotterdam on using our infrastructure to integrate more complex devices in Web interaction, such as a biometric pillow with embedded sensors for pressure, galvanic skin response (GSR), movement

patterns, and presence[14].

In future work we would like to explore and connect our infrastructure with other Web extension mechanisms, such as Internet Explorer plugins, or the Google Web Toolkit that enables building AJAX applications in the Java language[15]. We are also working on increasing the number of supported and tested OSS components. Our long term goal is to provide, as part of the open source distribution of AMICO:WEB, not only the platform itself, but also many wrapped OSS components, and/or auxiliary configuration files, that can be used directly with minimal configuration efforts. Our work is also connected and in many aspects complementary to recent W3C work on the Rich Web Application Backplane [2], which is intended to provide a more common integration infrastructure in the future.

9. ACKNOWLEDGEMENTS

Part of this research was funded by the European ITEA Passepartout project and the MultimediaN project of the BSIK programme of the Dutch Government, and by European Commission under contract FP6-027026 - Knowledge Space of semantic inference for automatic annotation and retrieval of multimedia content – K-Space. We thank Lynda Hardman who provided useful feedback on the work described here, and whose comments significantly improved this article.

10. REFERENCES

[1] C. Asakawa and T. Itoh. User interface of a Home Page Reader. In *ASSETS '98: Proceedings of the third international ACM conference on Assistive technologies*, pages 149–156, Marina del Rey, California, USA, 1998. ACM Press.

[2] M. Birbeck, J. Boyer, A. Gilman, K. Kelly, S. Pemberton, and C. Wiecha. Rich Web Application Backplane, W3C Note 19 July 2006, http://www.w3.org/MarkUp/Forms/2006/backplane/. W3C Note, 2006.

[3] M. Boyns. Muffin Web Filtering System, http://muffin.doit.org/, 2004.

[4] J. Clark. XSL Transformations (XSLT) Version 1.0. W3C Recommendation, 16 November 1999.

[5] ConceptNet. Commonsense and NLP Toolkit, http://web.media.mit.edu/~hugo/conceptnet/, 2006.

[6] W. Consortium. Extensible 3D (X3D) Specification (ISO/IEC 19775:2004), http://www.web3d.org/x3d/specifications/, 2004.

[7] W. K. Edwards. Putting computing in context: An infrastructure to support extensible context-enhanced collaborative applications. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 12(4):446–474, December 2005.

[8] FreeTTS. Text-to-Speech Synthesizer, http://freetts.sourceforge.net/, 2006.

[9] HandVu. Hand Gesture Recognitizer, http://www.movesinstitute.org/~kolsch/HandVu/, 2006.

[10] V. L. Hanson, J. P. Brezin, S. Crayne, S. Keates, R. Kjeldsen, J. T. Richards, C. Swart, and S. Trewin. Improving Web accessibility through an enhanced open-source browser. *IBM Systems Journal*, 44(3):573–588, October-December 2005.

[11] M. J. Karels. Commercializing Open Source Software. *ACM Queue*, 1(5), July–August 2003.

[12] A. Liu and I. Gorton. Accelerating COTS Middleware Acquisition: The i-Mate Process. *IEEE Software*, 20(1):72–79, January 2003.

[13] T. R. Madanmohan and R. De. Open Source Reuse in Commercial Firms. *IEEE Software*, 21(6):62–69, November 2004.

[14] MARY. Text-to-Speech System for English, German and Tibetan, http://mary.dfki.de/, 2006.

[15] R. C. Miller and B. A. Myers. Integrating a Command Shell into a Web Browser. In *Proceedings of USENIX 2000 Annual Technical Conference*, pages 171–182, San Diego, CA, 2000.

[16] S. Milne, A. Dickinson, A. Carmichael, D. Sloan, R. Eisma, and P. Gregor. Are guidelines enough? An introduction to designing Web sites accessible to older people. *IBM Systems Journal*, 44(3):557–572, October-December 2005.

[17] NeXTeNS. Text-to-Speech Synthesizer for Dutch, http://nextens.uvt.nl/, 2006.

[18] S. Oaks. *Java Security, 2 edition*. O'Reilly Media, 2001.

[19] Z. Obrenovic, F. Nack, and L. Hardman. Designing interactive ambient multimedia applications: requirements and implementation challenges. Technical Report INS-E0605, CWI, 2006.

[20] OpenCV. Computer Vision Library, http://opencvlibrary.sourceforge.net/, 2006.

[21] PAW. Pro-active Web Filter, http://paw-project.sourceforge.net/, 2006.

[22] J. T. Richards and V. L. Hanson. Web accessibility: a broader view. In *WWW '04: Proceedings of the 13th international Conference on World Wide Web*, pages 72–79, New York, NY, USA, 2004. ACM Press.

[23] Sesame. RDF database, http://www.openrdf.org/, 2006.

[24] Sphinx-4. English Speech Recognizer, http://cmusphinx.sourceforge.net/sphinx4/, 2006.

[25] W3C. Device Independence Activity. W3C working group.

[26] W3C. Web Accessibility Initiative. W3C working group.

[27] W3C. Multimodal Interaction Framework, W3C note, 6 May 2003, http://www.w3.org/TR/mmi-framework/, 2003.

[28] W3C. Alternative Web Browsing Index, http://www.w3.org/WAI/References/Browsing, 2006.

[29] W3C. "Voice Browser" Activity, http://www.w3.org/Voice/, 2006.

[30] WordNet. WordNet Lexical Reference System Project Page, http://wordnet.princeton.edu/, 2006.

[31] Zajicek, M., Powell, C., Reeves, and C. 1998. A Web navigation tool for the blind. In *ASSETS '98: Proceedings of the Third international ACM Conference on Assistive Technologies*, pages 204–206, Marina del Rey, California, USA, 1998. ACM Press.

[14] `http://www.defekt.nl/~moveme/`

[15] `http://code.google.com/webtoolkit/`