Modern Web APIs and Accessibility

Exploring the Text Track API, the Speech Input API, the Web Speech API and the CSS Speech Module.

Second Bachelor Thesis

Media Technology degree course
St. Pölten University of Applied Sciences

Completed by:
Francesco Novy
mt101064

Under the supervision of: Dipl.-Ing. (FH) Klaus Temper

St. Pölten, 02.06.2013

Declaration

- The attached research paper is my own, original work undertaken in partial fulfillment of my degree.

- I have made no use of sources, materials or assistance other than those which have been openly and fully acknowledged in the text. If any part of another person's work has been quoted, this either appears in inverted commas or (if beyond a few lines) is indented.

- Any direct quotation or source of ideas has been identified in the text by author, date, and page number(s) immediately after such an item, and full details are provided in a reference list at the end of the text.

- I understand that any breach of the fair practice regulations may result in a mark of zero for this research paper and that it could also involve other repercussions.

....	....
St. Pölten, on	Signature Author

Abstract

While new web technologies are being developed and proposed nearly daily, only a small part of them has the potential to create new possibilities for physically or mentally disabled people to explore the internet. This paper takes a few selected technologies which could possibly be used for this purpose and examines them for practicability, ease of use, functionality and availability. In addition, the general state of accessibility in web development will be shortly analysed, with a special focus on WAI-ARIA. The modern web APIs that will be explored are the Text Track API, the Speech Input API, the Web Speech API and the CSS Speech Module.

At the end of this thesis, readers will have an overview of these technologies, their possibilities and their shortfalls. The Text Track API is already relatively widely supported and works well. The Speech Input API is very easy to use but has no real prospect of widespread browser support. The Web Speech API is very powerful and, although it is at the moment only supported by Google Chrome, several browser vendors have announced plans for future support. Finally, the CSS Speech Modules meets a lot of criticism and is currently not supported by any major web browser.

Kurzfassung

Während fast täglich neue Webtechnologien entwickelt oder vorgeschlagen werden, haben nur sehr wenige davon das Potential das Webbrowsen für psychisch oder körperlich beeinträchtigte Personen positiv zu beeinflussen. Diese Arbeit nimmt einige ausgewählte Technologien, die möglicherweise für diesen Zweck genutzt werden können, und untersucht sie auf Praktikabilität, Funktionalität und Verfügbarkeit. Zusätzlich dazu werden aktuelle Vorgänge und Standards bezüglich Barrierefreiheit im Webdevelopment durchleuchtet. Die untersuchten Web Schnittstellen sind die Text Track API, die Speech Input API, die Web Speech API und das CSS Speech Module.

Am Ende dieser Arbeit werden Leser einen Überblick über diese Technologien, ihre Funktionalität und ihre Schwächen erhalten haben. Die Text Track API wird bereits von relativ vielen Browsern unterstützt und funktioniert bereits gut. Die Speech Input API ist sehr einfach zu nutzen, hat jedoch keine besonders guten Aussichten auf breiteren Browsersupport. Die Web Speech API ist sehr vielfältig und, obwohl sie im Moment nur in Google Chrome unterstützt wird, haben doch einige Browserhersteller bereits zukünftigen Support angekündigt. Das CSS Speech Module wird von vielen Experten kritisiert und funktioniert derzeit in keinem der meistgenutzten Webbrowser.

Table of Contents

1 Introduction

The World Wide Web (WWW) is one of the most important innovations of the last 50 years: only few would dispute this statement. According to a study by Cisco Systems from the end of 2011, which surveyed over 8000 college students and IT professionals under the age of 30 in 14 countries, over half of the interviewees said they could not live without the internet (Spector, 2011). The WWW is also a place of constant change: From January to end of May in 2013, 160 drafts and specifications have been introduced or updated by the W3C (W3C, 2013).

While these changes have brought richer internet applications, they have also bestowed an increased responsibility to provide ways and means for people with physical or mental disabilities to use them. Accessing textual information works considerably well with modern aids like screen readers, but working with media can still be a chore for users with disabilities.

Many of the newly introduced web technologies provide increased complexity and possibilities to interested web developers. But are there also new ways to create richer applications for people with physical of mental impairments?

This is the starting point for this work. The hypothesis is that there are many new or emerging techniques out there that could be used to provide enhanced functionality for people with disabilities. This thesis sets out to explore modern HTML and JavaScript APIs (Application Programming Interfaces) that can be used to work with accessible web design. For this purpose, four technologies have been selected: The Text Track API, the Speech Input API, the Web Speech API and the CSS Speech Module. These APIs will be examined on their possibilities and features. For this purpose, the available specifications will be analysed. Another major part of this paper is the investigation of current web browser support for these technologies: Which features can currently be used and in which browsers will they work? This knowledge is vital for web developers who want to implement one of these APIs.

While the main focus of this work lies on practicability and available components, future prospects for features and support will be given where possible. A test suite has also been devised, which can be found at http://www.fnovy.com/bac2 or appended to this work. This suite enables easy testing for browser support of future browser versions: Because versions change often and new features get introduced or revised at a fast pace, support for these APIs can change fast. The test suite provides an easy way to test on feature support, now or in the future: by visiting the suite with a browser of choise, this browser’s feature support can quickly be determined.

The central research questions this paper wants to answer are:

Which new HTML5 and JavaScript APIs can be used to increase the accessibility of modern web apps? How can they be used in real world scenarios and which web browsers support them?

This thesis will be separated in five main sections. First, a short introduction to the topic of accessibility in web development will be given. A special focus will be laid on WAI-ARIA (Web Accessibility Initiative – Accessible Rich Internet Applications), and general problems will be explored.

In section 2 (chapter 3), the Text Track API will be examined. Text Tracks can be added to multimedial elements like videos to provide built-in subtitles or other features. An overview of the API will be given, including attributes, methods and events. Then, web browser support will be discussed. For this purpose, various desktop and mobile browsers have been tested for basic and multilingual Text Track Support.

Section 3 (chapter 4) will analyse the Speech Input API. This API adds an easy way to enable voice input for HTML input fields. Available attributes and methods will be explored, followed by a short browser support test for Speech Input and its variations.

The next section will cover the Web Speech API, which can be considered as an advanced and more complex version of the Speech Input API. It provides powerful functionality to work with continual speech recognition, which can be used in manifold ways. Again, an introduction to the API and its functions will be given first, followed by an example of how it could be used in a real world scenario. Following this, a short speech recognition accuracy test will be included, which will give a rough overview of how well speech recognition currently works. Speech synthesis, another part of the Web Speech API which has not yet been implemented by any browser vendor, will also be introduced briefly. Browser support and support prospects will also be discussed.

The final section (chapter 6) will give a quick overview of the CSS Speech Module, which allows web developers to include aural style sheets to their websites.

2 State of Rich Internet Application accessibility

While the importance of the World Wide Web is steadily rising, the topic of web accessibility is also gaining web developers' mind share. Nevertheless browsing websites which rely on anything but static text can still be a chore for people with visual, auditorial, mental or other physical disabilities.

There are of course several ways to increase a website's accessibility. The possibly most important initiative in this field is the Web Accessibility Initiative (WAI) by the World Wide Web Consortium (W3C). This chapter will give a short overview of the W3C's efforts, as well as a few general topics concerning accessibility and modern web technologies.

2.1 WAI-ARIA

WAI-ARIA stands for "Web Accessibility Initiative – Accessible Rich Internet Applications" and is "(…) a technical specification that provides a framework to improve the accessibility and interoperability of web content and applications" (James & Cooper, 2011). It is currently being developed by the World Wide Web Consortium (W3C) and declared as of January 2011 as a Candidate Recommendation, which means that it has been widely reviewed and is near to its final form (Jacobs, 2005). Originally designed as a XML Name Space, it was later decided to use HTML attributes instead because most web developers were not comfortable with XML (Thiessen, 2011).

WAI-ARIA is a very detailed specification, but fundamentally it gives websites the possibility to declare themselves as interactive applications rather than static documents. It allows developers to add additional semantic information to an HTML document in order to make dynamic content more accessible. For this purpose, WAI-ARIA introduces "Live Regions". The details specified in the following chapter have been compiled from the W3C WAI-ARIA 1.0 Candidate Recommendation. (James & Cooper, 2011, p. -)

The WAI-ARIA Primer (Pappas, Schwerdtfeger, & Cooper, 2010), an accompanying document to the actual specification, acts as a technical introduction to the topic and describes the problems that WAI-ARIA is supposed to solve. It also explores the underlying concepts of web accessibility as well as providing reasons for web developers for adopting WAI-ARIA.

These reasons are broadly separated in two categories: technical and business-related reasons. On the technical side, WAI-ARIA allows developers to stay WCAG 2.0 (Caldwell, Cooper, Reid, & Vanderheiden, 2008) instead of WCAG 1.0 (Chisholm, Vanderheiden, & Jacobs, 1999) compliant. WCAG 1.0 required that content can be displayed with all scripts and Style Sheets turned off. For many modern websites, while deactivating everything might still show information that can be processed, it would also result in a much more unenjoyable experience compared to the script-enhanced and styled version.

From a business point of view, the Primer promotes WAI-ARIA's benefits for all users, not just the ones with disabilities. According to the Primer, WAI-ARIA establishes a web standard for keyboard interactions and facilitates test automation by providing the needed semantic information.

The Draft "Using WAI-ARIA in HTML" provided by the W3C (Faulkner, 2013) contains further practical aid for developers who want to apply WAI-ARIA attributes to their projects.

2.1.1 Live Regions

Modern web applications can often contain a lot of dynamically changing content. For example, a news ticker might display content that changes every few seconds. Because people who rely on screen readers or similar to browse the web can only access information in a sequential order, it is difficult for them to deal with such dynamic changes: screen readers have no way of knowing if the updated content is important or not and how it should be dealt with.

Because of this, WAI-ARIA proposes so called Live Regions. Live Regions are "(…) perceivable regions of a web page that are typically updated as a result of an external event when user focus may be elsewhere." (James & Cooper, 2011)

While Live Regions are only a small subset of WAI-ARIA, the author of this paper believes that it is the most basic and most important feature that WAI-ARIA specifies, because it solves one of the most pressing problems that modern Rich Internet Applications (RIAs) provide for people with disabilities: the problem of merely knowing of dynamic content changes. Because of this, Live Regions will be shortly explained in this chapter. For further information about Live Regions or other WAI-ARIA attributes, the reading of the WAI-ARIA guidelines is recommended to all web developers (James & Cooper, 2011).

The basic attribute is `aria-live`. By default, all elements have `aria-live` set to `off`, which means that updates to this element will not be presented to assistive technologies like screen readers unless the user is currently focused on that element.

Other possible values are:

- `polite` is used for background changes; Assistive technologies should announce updates at the next opportunity like for example the end of the current sentence.
- `assertive` is used to tell assistive technologies to notify the user immediately. Because this may lead to disorientation of the user, `assertive` should only be used when an immediate notification is really necessary.

With the `aria-atomic` property, it is possible to determine if, when a change occurs, the user should be notified only of the changed content or if the changed content should be presented in the context of a parent element. By default, all elements have `aria-atomic` set to `false`.

Another related attribute is `aria-relevant`: It tells assistive technologies what type of content changes should be regarded for a specific live region. An element can have one or multiple values as a space delimited list, where the allowed values are:

- `additions` triggers if nodes are added to the element.
- `removals` triggers if nodes are removed from the element.
- `text` triggers if the textual content of the node or its children changes.
- `all` for a combination of the above.

The default value is `additions text`.

An additional property is `aria-busy`, which should be added to an element if the element or its subtree is currently being updated. If the update has finished, `aria-busy` should be set to `false` to indicate that the update is complete.

2.1.2 Problems with WAI-ARIA

After studying the detailed specification by the W3C, the main question that remains is: Why has WAI-ARIA not reached widespread adoption yet?

ARIA would, theoretically, be an ideal solution to give screen readers and other assistive technology the ability to deal with dynamic content. However, ARIA is only a specification that describes how something should be implemented. It is completely up to browser vendors and web developers to actually implement any of WAI-ARIA’s standards.

Most of the dynamic content available today does not follow the ARIA guidelines at all (Borodin, Bigham, Dausch, & Ramakrishnan, 2010). The main reason for this is that it would take considerable additional work and engagement from developers. To check if a website uses valid ARIA, the W3C provides the NU Markup Validation Service (W3C, n.d.).

2.2 Multimedia

Native web multimedia elements include, in the current specification of the W3C, audio and video (Berjon, Faulkner, Navara, O'Connor, & Pfeiffer, 2013). Web browsers that support these elements do not have to rely on external plugins anymore to play these types of media.

For example, the most common way to provide a video was to include it as an Adobe Flash file. Adobe provides Accessibility best practices and various ways to increase accessibility, like the ability to add closed captions or accessible video control skins (Adobe, n.d.).

Web developers who want to use the newest technologies and who do not want to be forced to rely on external plugins have to replace this functions that dedicated solutions provide with comparable features. For this purpose, the HTML5 specification contains new features like Timed Text Tracks, which will be further evaluated in later chapters of this work.

2.3 JavaScript libraries

Although there is a huge number of JavaScript libraries to chose from, usage statistics show that an overwhelming majority of users use only a small subset of these libraries.

According to W3Techs, which is dedicated to web technology surveys, on April 25th 2013 62.6% of all checked websites use at least one JavaScript library (W3Techs, 2013). Of these, an overwhelming 91.5% of webistes use jQuery. On a distant second place is MooTools with 7.6%, followed by Prototype with 5.5%.

As of now, the three major libraries jQuery (version 2.0.0), MooTools (version 1.4.5) and Prototype (version 1.7.1) have no features concerning accessibility, as far as the libraries' documentations show.

2.4 Common accessibility problems of modern web applications

There are a few problems that frequently arise in web development regarding accessibility. Many of these problems can be easily avoided. This chapter will look into some of these common problems. The following points have been compiled from the Yahoo Accessibility Blog (Kloots, 2011).

Show the current focus

Many designers reset the default browser stylings for focused elements. While this is done for aesthetical reasons, it can seriously impact accessibility because it might not be evident where a user currently is when tabbing through a document. This can easily be corrected by applying corresponding styles in your Cascading Style Sheets (CSS), like this:

```
a:focus {
   outline: dotted 1px #000;
}
```

Listing 1: Adding focus styling to elements

Don't use hyperlinks as buttons

Hyperlinks are often used as buttons. While it makes no difference for sighted users if an interactive element is (semantically) an `a`-element or a button, a screen reader will assume that it is a hyperlink. Often, the `href`-attribute is set to "#" or "javascript: …", which assertive technology will often simply read out loud. To solve this, developers can either add a `button` role to the hyperlink, or replace the hyperlink with a `button` element.

Use labels for UI controls

UI controls should always be accompanied by corresponding labels. Screen readers will announce a control's label when the control receives focus, and if the label is missing the user will not be aware of the control's function or purpose.

Use semantic attributes for input fields

In addition to a visual indication like an asterisk before or after the field's label, an input field should always have a `required` attribute as specified in HTML 5 (Berjon et al., 2013). Also, if a field is validated and deemed invalid, developers should add an `aria-invalid` attribute to the input field.

3 Text Track API

As specified in HTML5, a media element (meaning an audio or a video element) can have a group of corresponding text tracks. Text tracks can represent various types of information:

- Subtitles
- Captions
- Descriptions
- Chapters
- Metadata

Each track element has a `src` attribute that points to a text file which contains data in the form of timed track cues, which can theoretically be in any format a browser can parse. Currently, WebVTT (Web Video Text Tracks) is supported by recent versions of Google Chrome, Safari, Opera and Internet Explorer 10, while the TTML (Timed Text Markup Language) format is only supported by Internet Explorer 10. Mozilla Firefox has not implemented Text Tracks as of yet (Pfeiffer, 2012). There are however several small JavaScript polyfills like Captionator ("Captionator," n.d.) which supplement support for Text Tracks to all browsers that are capable of displaying HTML5 video. While scripts like this provide support for several Timed Text Track formats, this work will concentrate on WebVTT, which is the natively most widely supported format.

The specifications detailed in the following chapters have been taken from the most current W3C guideline regarding HTML5, the HTML5.1 Nightly Editors Draft (Berjon et al., 2013).

3.1 Text Track API overview

There are different possibilities to use text tracks. The simplest one is to include one or more `<track>` elements to a media element:

```
<video src="video.ogg">
   <track kind="subtitles" srclang="en" label="English Subtitles"
default src="subtitles_en.vtt"></track>
</video>
```

Listing 2: A simple example for the inclusion of a TextTrack element.

A `<track>` element has various possible attributes:

- `kind`: This decides how the web browser handles the text track. However, this depends on web browsers' implementation.
- `label`: A human-readable text that should describe the track to the user, e.g. "English subtitles".
- `srclang`: The language of the text track, e.g. "en". Browsers should allow users to switch between different languages if multiple text tracks are provided.
- `src`: The address of the source file.
- `default`: if the track should be displayed by default.

These attributes can be added directly in the HTML DOM.

3.2 Text Track JavaScript API

In addition to the HTML attributes, there is a JavaScript API that provides various functions to work with text tracks. The most important ones will be described below.

To create a new `TextTrack` object and add it to the media element, each media- (e.g. video-) element provides an `addTextTrack`-function:

```
media.addTextTrack(kind, label, language)
```

This will create and return a new TextTrack with the specified parameters.

TextTrack objects provide various methods:

```
textTrack.kind / textTrack.label / textTrack.language
```

This returns the corresponding attribute of the text track.

```
textTrack.readyState
```

`readyState` returns the track's readiness state. This can be one of the following:

- `0` (none): The track has not been loaded yet.
- `1` (loading): The track is currently being loaded.
- `2` (loaded): The track has been loaded successfully.
- `3` (error): The text track has failed to load.

`textTrack.mode [=value]`

This returns the mode of the text track. This can be one of the following:

- "`showing`": The text track is currently being displayed.
- "`hidden`": The text track is active but currently not displayed be the browser.
- "`disabled`": The text track is disabled.

Via `textTrack.mode = "value"` the mode can be changed.

`textTrack.cues / textTrack.activeCues`

This returns the list of cues of the `TextTrack` object as a `TextTrackCueList` object. `activeCues` will only return the cues that are currently displayed or that follow after the current cue.

`textTrack.addCue(cue) / textTrack.removeCue(cue)`

Adds or removes a cue from the `TextTrack` object.

A `TextTrackCueList` object represents the list of text track cues in a given order, which enables developers to get individual text cues. It provides the following features:

`cuelist.length`: Returns the number of cues in the TextTrackCueList.

`cuelist[index]`: Returns the cue at the specified position in the cue list.

`cuelist.getCueById(id)`: Returns the cue with the given id or null if no cue with the specified id is found.

The smallest entity regarding the Text Track API is `TextTrackCue`. It has the following attributes:

`cue.track`: Returns the `TextTrack` object to which this cue belongs or null if it belongs to no `TextTrack` object.

`cue.id [=value]`: Returns the cue's id, and can also be set by applying a value.

`cue.startTime [=value] / cue.endTime [=value]`: Returns the start or end time in seconds and can also be set by applying values.

`cue.pauseOnExit [=value]`: Returns if the media should be pause when the cue is finished and can also be set by applying a value.

3.3 Text Track web browser support

While the specification by the W3C is quite advanced, implementation by web browsers is still lacking behind. The main problem, however, lies in the fact that it is very hard to actually find out which browser supports which features, and therefore which features actually can be used in real world scenarios.

For this purpose, a small testing suite has been developed and deployed on http://www.fnovy.com/bac2/texttrack to simplify browser support testing. The suite consists of two tests that can be opened in a web browser. Each test specifies which result a browser should provide. By comparing the result in a browser with the stated "expected outcome", it is possible to find out which features are currently supported.

A preliminary step was to find out which browsers support HTML5 video in general. This has been done to find out which platforms can safely be excluded from all further tests regarding TextTrack support. For this purpose, a video that has been published under the Creative Commons License has been converted in the video formats that are currently most widely supported, namely h.264 (.mp4), WebM (.webm) and Theora (.ogg) and included in a simple HTML page (Accessibility in Blended Learning, 2012). All further tests are built on top of this basic setup.

Support for HTML5 video is in general very good. All tested browsers have support for at least one of the used video formats, except for Internet Explorer 8 and Opera Mini. Also, in Safari 5 on Windows, no video has been displayed, although according to Apple it should actually support HTML5 video (Apple, n.d.). This means that, according to the statistical data by StatCounter gathered in the time span January to March 2013, over 80% of users use HTML5 video enabled desktop browsers (StatCounter, 2013).

The first test regarding Text Track was for basic WebVTT support. For this purpose, a simple WebVTT file containing sub titles for the video has been created. Only the most basic properties for WebVTT have been used in this file: raw text and time markers.

```
00:11.303 --> 00:17.302
As a lector and teacher in the philosophy department,
most of my classes are face to face classes, interal classes.
```

Listing 3: Basic WebVTT example

Then, this single WebVTT has been added via a `track`-element to the video-element.

```
<video controls="controls" preload="preload" title="Accessibility in
Blended Learning">
   <source src="res/video.webm" type="video/webm" />
   <source src="res/video.ogv" type="video/ogg" />
   <source src="res/video.mp4" type="video/mp4" />

   <track kind="subtitles" label="English subtitles" src="
subtitles_en.vtt" srclang="en" default></track>
</video>
```

Listing 4: Multiple sources as well as one track element have been added to the video element.

The first test is passed by Chrome (and subsequently by Chrome Canary) as well as by Opera and Safari 6 (which is only available for Mac OS X). However, Opera and Safari 6 failed to provide any way to toggle the display of the subtitles. At first, the subtitles were not displayed by Internet Explorer 10, although other tests have allocated this capability to it. After some research, it turned out that Internet Explorer 10 is simply very strict regarding the encoding of WebVTT-files. The .vtt file has to be encoded in UTF-8, and on the server-side the MIME-type for .vtt-files has to be set. The easiest way to do this is via the .htaccess command `AddType text/vtt vtt`.

After these steps, Internet Explorer 10 displayed the subtitles correctly, too.

On mobile browsers, there is less support for this basic WebVTT setup. On Chrome for Android, the subtitles are actually displayed, but they are not displayed correctly. The latest Opera Mobile Beta 14, which is also based on Webkit, has exactly the same display bug. It is apparently the correct text and it changes at the right moments, but unfortunately it is unreadable. However, in the current beta release (version 27), text tracks actually work correctly. As far as tests have shown, this is the only mobile browser which supports them at the moment – all other browsers fail to display anything.

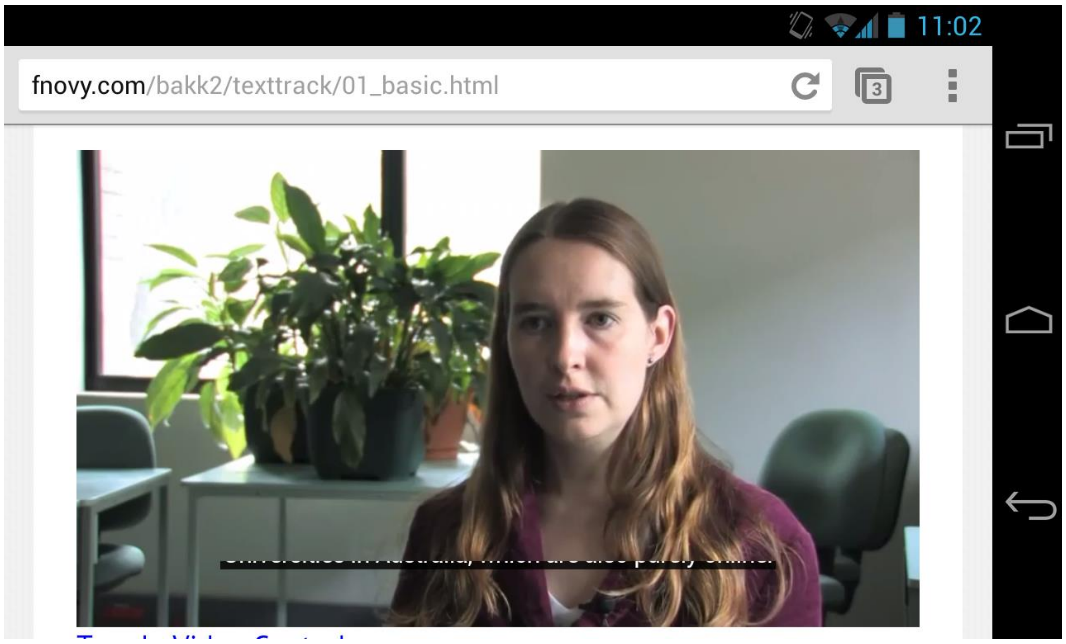

Figure 1: On Chrome 26 and Opera 14 Beta for Android there is a display bug.

The second test was for multi-language support. This feature in particular can be very useful for sighted users, too, because they can get in-time translation for the media they are consuming. For the test, two tracks were included: an English one and a German one.

The results for these tests were much worse. Only Internet Explorer 10 displayed the subtitles correctly: A switch was present to select one of the multiple text tracks, named correctly with the specified labels.

In Opera and Safari, again there was not even a toggle present to disable or enable text tracks. Therefore, there was no way to change between different tracks.

In Chrome, the Stable Release (version 26) and the Canary Release (version 28) displayed different behaviour – but both had no way to toggle between the different languages. In Chrome 28, the second, German track was displayed, although the English one, which came first in the HTML, had the `default` attribute. In Chrome 26, the behaviour was even different: Although by default it displayed the (correct) English subtitles, if the subtitles were disabled with the corresponding button, and enabled again, suddenly both tracks were displayed. This could not be reproduced in Chrome Canary version 28.

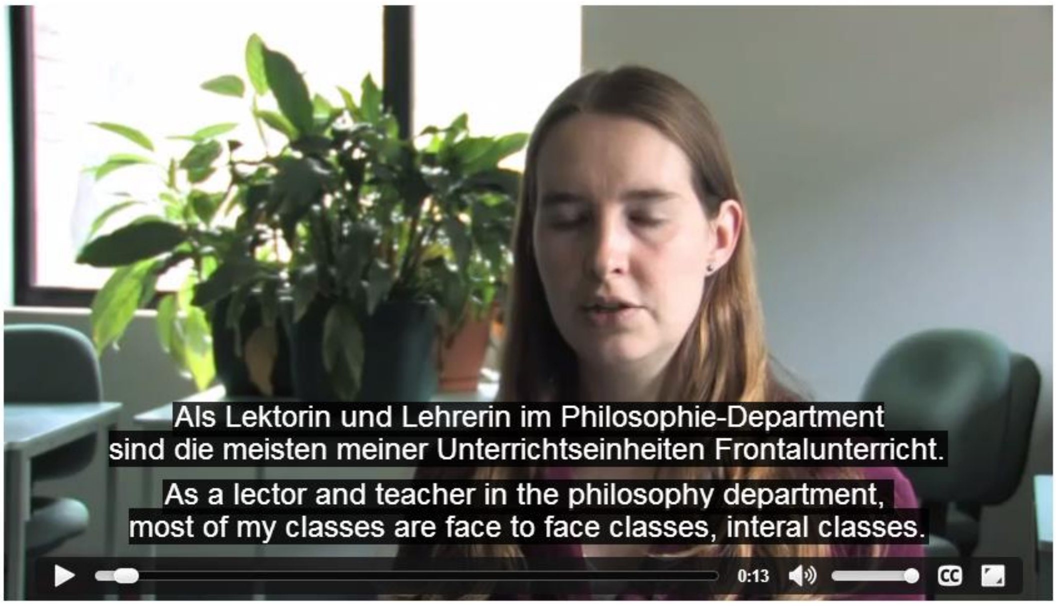
Als Lektorin und Lehrerin im Philosophie-Department
sind die meisten meiner Unterrichtseinheiten Frontalunterricht.
As a lector and teacher in the philosophy department,
most of my classes are face to face classes, interal classes.
0:13

Figure 2: Chrome 26 displays both tracks, when they are disabled and enabled again.

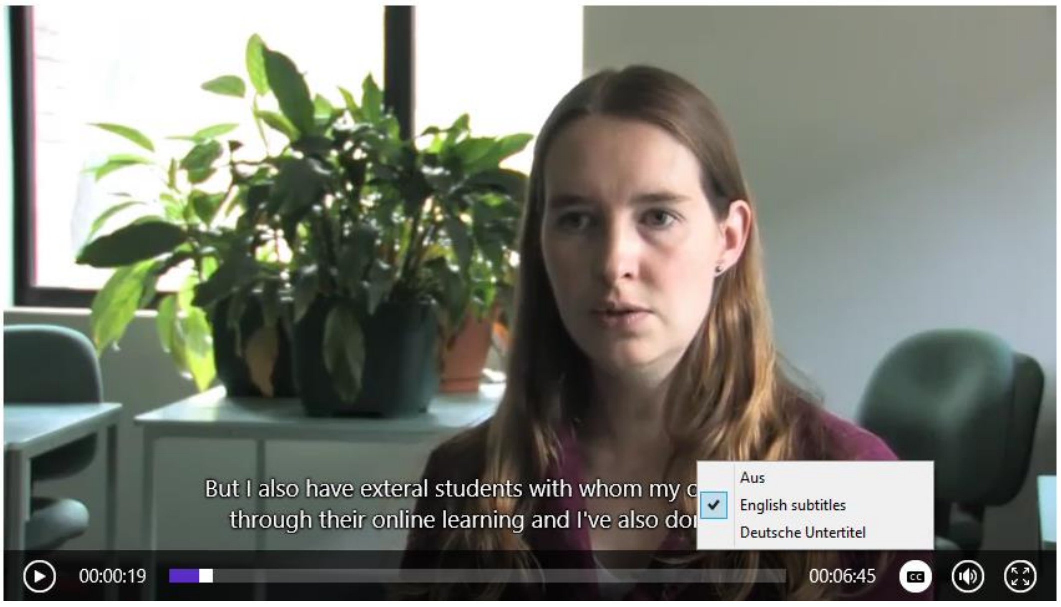
But I also have exteral students with whom my c
through their online learning and I've also do
Aus
✓ English subtitles
Deutsche Untertitel
00:00:19
00:06:45

Figure 3: Only Internet Explorer 10 displayed the track selector correctly.

All in all, at the moment multiple tracks per media element should be avoided. Not only will only Internet Explorer 10 users be able to chose between the tracks, but for Chrome users the experience gets actually worse than if only one track was present. It can, however, be hoped that these bugs will be sorted out soon. Until then, single text tracks can be used safely, because browsers with no support will simply display the video normally without any text tracks.

Below, two tables have been compiled for desktop and mobile browsers. In these tables all test results and the tested browser versions can be found.

	HTML5 Video	Basic WebVTT	Multi-language
Chrome 26	Yes	Yes	No
Chrome 28 Canary	Yes	Yes	No
Firefox 19	Yes	No	-
Firefox 21 Aurora	Yes	No	-
Internet Explorer 8	No	-	-
Internet Explorer 9	Yes	No	-
Internet Explorer 10	Yes	Yes[1]	Yes
Opera 12.15	Yes	Yes[2]	No
Safari 5 (Windows)	No	-	-
Safari 6 (Mac OS X)	Yes	Yes[2]	No

Table 1: TextTrack compatibility on desktop web browsers.

[1] The .vtt-file has to be encoded in UTF-8 and the server has to specify its MIME-type correctly.

[2] No toggle for disabling and enabling tracks was displayed.

	HTML5 Video	Basic WebVTT	Multi-language
Chrome 26	Yes	No[3]	-
Chrome 27 (Beta)	Yes	Yes	No
Firefox 20	Yes	No	-
Firefox 21 (Beta)	Yes	No	-
Opera Mobile 12.10	Yes	No	-
Opera Mini 7.5	No	-	-
Opera 14 (Beta)	Yes	No[3]	-
Dolphin Browser 9.4	Yes	No	-
Boat Browser 5.2	Yes	No	-
Safari 6 (iPhone)	Yes	No	-
Safari 6 (iPad)	Yes	No	-
Internet Explorer 10 (Mobile)	Yes	No	-

Table 2: TextTrack comptatibility on mobile web browsers.

[3] Subtitles are present, but not displayed correctly.

Of course, subtitles are only a subset of the possibilities that the TextTrack API provides. However, analysing all features of the API is out of the scope of this work and should be done in future work.

4 Speech Input API

Audio in web applications is becoming a more important topic because APIs for speech input or synthesis are slowly getting traction in recent times.

The Speech Input API, which has been proposed by Google, is currently only supported by Google Chrome (Sampath & Bringert, 2010). It has been added in April 2011 (Estelle, 2011). The Speech Input API is already a W3C Editor's Draft, which means that while it is already on its way to standardisation, it is still not quite final (Jacobs, 2005).

With the HTML5 Speech Input API, speech input functionality can easily be added to input fields. Developers simply have to add a `speech` attribute to their text fields. Currently, only Chrome supports this feature, but only if the experimental attribute `x-webkit-speech` is used. A working example would be:

```
<input type=”text” speech x-webkit-speech />
```

Listing 5: A simple Speech Input example.

According to the specification, the `speech` attribute can be added to all HTML input elements with one of the following types: text, search, url, telephone, email and password. It is explicitly noted that the usage of speech input for password fields might have security-related issues, but that it is ultimately up to the user to decide if they want to speak their password (Sampath & Bringert, 2010). The `speech` attribute (and browser-specific variations like `x-webkit-speech`) can safely be used because browsers that do not support them will simply ignore them and work like regular input fields.

An important point to understand is that the Speech Input API does not specify how the audio parsing should be done. It is simply an interface between a service that a browser vendor embeds and the user. In the case of Google Chrome, the captured audio gets send to Google's servers, which analyse the input and return the understood words to the API, which in turn transmits them to the user (Pultz, 2011).

While this will work just fine on its own (if tested in Google Chrome), there are also further, additional attributes that can be used to adjust the speech input. Some of these attributes will be shortly described below.

`maxresults=“value“`

This attribute sets the maximum number of items that should be placed in the captured sentence. This makes it possible to e.g. restrict speech input to a single word.

`nospeechtimeout=”value”`

Specifies the time in milliseconds the browser will wait for the speech to start. If no speech is recognized when this time span passed, audio capture will stop.

`capturestart=”function”`

This will fire an event when speech capture is started.

`speechstart=”function” / speechend=”function”`

This event will call a function when a new speech input is detected after audio capture has been started respectively when it is recognized that the user has stopped speaking.

`speechchange=”function”`

This event is dispatched when a set of valid speech inputs has been recognized.

`speecherror=”function”`

This will be called when the active speech input session resulted in an error.

Important to note is that speech input relies on the `lang` attribute. If an input field has a valid `lang` attribute, speech input will use this specified language. Else the language of the document will be used.

There are also a couple of JavaScript functions for manually starting or stopping speech input. They work like this:

```
document.getElementById(“fieldWithSpeechInput”).startSpeechInput();
document.getElementById(“fieldWithSpeechInput”).stopSpeechInput();
document.getElementById(“fieldWithSpeechInput”).cancelSpeechInput();
```

Listing 6: Speech Input JavaScript functions

The difference between `stopSpeechInput()` and `cancelSpeechInput()` is that while the former will process everything the user has said so far, the latter will abort instantly, discard any pending audio data and fire no events.

4.1 Speech Input API browser support

A small site for testing of the Speech Input API has been created at http://fnovy.com/bac2/audio/speechinput.html. By going through the predefined steps it is possible to find out how far browsers support the specification.

As mentioned above, Google Chrome is the only browser that currently supports Speech Input at all, and Chrome relies on vendor-prefixed versions of the attributes because the spec is still under development and subject to change.

At the moment, only basic speech capture is supported by Chrome. The attributes `nospeechtimeout` and `maxresults` are ignored, and the JavaScript functions are not supported at all.

The events do not fire correctly. Only the `onspeechchange` (more precisely, the prefixed version `onwebkitspeechchange`) works by firing after the text has been processed. By parsing the content of the respective input field, it is therefore possible to react to different speech input. Another working possibility is to parse the `onspeechchange`-event-object for alternate utterances that might have been detected.

```
// HTML:
<input type="text"speech onspeechchange="checkInput(event);" />

// JavaScript:
function checkInput(event) {
   var results = event.results;
   for(var i=0; i<results.length; i++) {
      var utterance = result[i].utterance;
      var confidence = result[i].confidence;
   }
}
```

Listing 7: Working with multiple results of the Speech Input API

5 Web Speech API

While all technologies discussed in this work are relatively new, the Web Speech API is even more current: In February 2013, Google has added support for the Web Speech API to Chrome 25 standard release, and it is since then the only browser with any level of support (Shires, 2013).

In general, the Web Speech API has been pushed more or less by Google alone: two Google engineers have introduced the Web Speech API Specification and proposed it to the W3C to eventually become a standard (Shires & Wennborg, 2012).

Despite its lack of support in other browsers, the Web Speech API is currently the major advance in the field of speech recognition and synthesis in mainstream web browsers. Because of Chrome's market share of over one third (StatCounter, 2013), it can nonetheless reach a considerable amount of users.

The main difference between the Web Speech API and the Speech Input API is that the Speech Input API has a more basic (and limited) functionality. The Web Speech API is much more advanced and offers more possibilities. While the Speech Input API is bound to HTML input fields and designed to handle short text inputs for a specified field, the Web Speech API can keep capturing audio for an infinite time and parse the results on the fly, without finishing the audio capture.

While Speech Input can be used without any kind of scripting, the Web Speech API is entirely JavaScript-based. Because it is currently only supported in Chrome, only the vendor prefixed versions with "webkit" are currently working. Additional details can be found in the W3C Web Speech API Community Group Final Report, which the following data is based on (Shires & Wennborg, 2012).

5.1 SpeechRecognition API overview

The main part of the Web Speech API is SpeechRecognition. It enables developers to parse audio streams that users say and get the parsed input as a string. The following chapters will give an overview of the specification.

5.1.1 SpeechRecognition Attributes

The first step is to initialize a `SpeechRecognition` object. This object provides the speech interface and has several attributes that can be set. These include:

`continuous=”value”`

If continuous is set to true (the default is false), the browser will continue speech recognition after a correct set of utterances is detected. With false, Web Speech works more like Speech Input, but when continuous is set to true, it is possible to keep recording and keep parsing input for the whole time a user is on a website.

`interimResults=”value”`

This attributes specifies if interim results should be returned. If this is set to true, the user agent will return utterances where it is not yet completely sure that they are correct. Interim results can be discerned by final results by checking their `isFinal` attribute.

`maxAlternatives=”value”`

Controls the maximum number of recognized alternatives per result. The default value for this attribute is 1.

`serviceURI=”value”`

This is a very powerful attribute that is another potent differentiator to the Speech Input API. It allows the developer to specify an URL that points to the location of a speech recognition service that should be used. Only if this attribute is unset at the time the speech input is started the browser will use the default service that the browser vendor included (in Chrome’s case, a Google service). `serviceURI` can also point to local services. The draft also mentions that it should be possible fot the end user to change the default service used in the browser configuration.

This addresses a problem that the Speech Input API faced: developers (and users) who want to use speech input rely on an external server which gets the data of everything a user inputs. For some people, this might be a severe privacy intrusion, especially if it is not even possible to specify which service should process the data. By being able to change this manually, especially websites with high privacy guidelines like for example banks could possibly use their own, secure speech recognition service.

However, as far as the tests conducted for this work have shown, the current implementation in Google Chrome has no support for the serviceURI attribute – it will always use Google’s speech recognition service.

An example for the initialisation of speech recognition could be the following:

```
var recognition = new SpeechRecognition();
recognition.continuous = true;
recognition.interimResults = false;
recognition.serviceURI = "http://yourservice.com/speech-api";
```

Listing 8: Initialisation of speech recognition

5.1.2 SpeechRecognition Methods

The `SpeechRecognition` object has only three methods to call on: `start()`, `stop()` and `abort()`. `start()` tells the user agent that the service should begin to listen and try to recognize the speech. `stop()` and `abort()` both will stop listening, but `stop()` will try to return a result using the audio it has already received, while `abort()` will stop immediately and return nothing. Both methods will fire the `end` (or `error`) event.

5.1.3 SpeechRecognition Events

This API provides various events that can be used to work with the speech input.

`audiostart / audioend`

This is fired when the user agent has started to capture audio or when it has finished capturing audio.

`soundstart / soundend`

This is fired when some sound, which can be speech but does not have to be speech, has been detected or when no sound is detected anymore.

`speechstart / speechend`

This event is called when speech has been recognized or when speech that will be used for speech recognition has ended.

`start / end`

This gets fired when the speech recognition service has begun listening to the audio stream or when it has disconnected. `end` will always be fired when the session ends, no matter the reason.

`result`

Will be fired when the speech recognizer returns a result.

`nomatch`

This event is fired when a final result is returned with no recognized utterance that is probable enough. Note that this event may still contain results, but they will be below the confidence threshold or may be null.

`error`

This is fired when a speech recognition error occurs. This could e.g. happen if the client could not connect to the specified `serviceURI`.

5.2 Basic usage of SpeechRecognition

A simple usage example for speech recognition would be to allow dictation of a text. To realise this, a new `SpeechRecognition` object has to be initialised and the according events have to be set.

```
var recognition = new SpeechRecognition();
recognition.continuous = true; // We want to keep recording
recognition.lang = "en_UK";
var isRecording = false;

recognition.onstart = function() {
   isRecording = true;
}
recognition.onresult = checkResult;
recognition.onerror = function(event) {/* show error message */}
recognition.onend = function() {
   isRecording = false;
}
```

Listing 9: Initialising speech recognition and setting the appropriate event handlers

This code snippet initialises a new `SpeechRecognition` object and sets its continuous attribute to true. The variable `isRecording` can be used to find out if speech is currently being recorded. When speech has been detected, the method `checkResults(event)` will be called. This function handles the speech input:

```
function checkResult(event) {
      var transcript = "";
      for(var i=0; i<event.results.length; i++) {
         transcript += event.results[i][0].transcript;
      }
      // Do something with transcript
   }
```

Listing 10: The function which handles the recognised input

This will enable very simple speech recognition! The completed example with start and end buttons can be found at:

http://www.fnovy.com/bac2/audio/webspeech.html

Note that because this API is still experimental, implementation is still vendor prefixed. To make implementation as future proof as possible, all possible major vendor prefixes should be accounted for. In addition, it should always be checked if the browser supports the feature. An easy way to achieve this would be:

```
var SpeechRecognition = window.SpeechRecognition ||
      window.webkitSpeechRecognition ||
      window.mozSpeechRecognition ||
      window.oSpeechRecognition ||
      window.msSpeechRecognition;

if (!SpeechRecognition) {
   // browser has no support!
} else {
   // This will initialise the supported prefixed version
   var recognition = new SpeechRecognition();
   // work with recognition
}
```

Listing 11: Working with vendor-prefixed versions of SpeechRecognition

5.3 SpeechRecognition accuracy

At the moment, only Chrome has implemented the Web Speech API. While the implementation is relatively simple and works well, the usefulness of the Web Speech API depends heavily on the quality of the speech recognition.

For this purpose, a small test has been conducted. Seven users have been tasked with reading a predefined list of words and sentences. Then, it has been checked if the recognized speech input matched the words that were predefined. The results have been compared and analysed to provide a basic overview of how good speech input works. Note that this test has not been exhaustive and does not claim to be completely comprehensive, but only to give a rough impression of the current state of speech recognition. The result of this test is primarily a test of the given Google web service and not of the Web Speech API itself. Nonetheless, for the end user and therefore for the developer it does not matter that the result is not directly dependant on the Web Speech API, because the API’s usefulness is directly linked to the quality of the integrated web service.

The test consists of 8 simple words, 4 short phrases and 4 short sentences. The complete list of tested texts can be found in table 4.

In general, the speech recognition worked considerably well. In total, 76% of the read texts were recognised. In the test, simple sentences had the best detection rate, with 93% recognised sentences. In comparison, single words were detected correctly 75% of the time, while short phrases could only reach a 64% detection rate.

Another important criterium that has been tested was the time it took between the users reading the text and the speech recognition service returning a result. In total, it took an average of 7.70 seconds for the users to either get a correct result or give up and move on to the next word because recognition has not worked. Notably, it took at least three seconds to return a result. It seems likely that this is the time that the Google speech recognition service needs to get, check and return a result for an audio stream. Therefore, at the moment, developers should always take this three second delay into account when working with speech recognition. It should be noted that the participants have taken the test from their homes, which undermines the assumption that the delay is caused by the speech recognition service and not by the user’s internet connection. Further tests should be done to prove this assumption.

It has to be noted that two native English speakers and five people with English as their first foreign language have participated in this test. While the native speakers had very good results, there were also non-native speakers with comparably good results. All test participants have self-rated their English proficiency as good or very good.

As a conclusion for this test, the following points could be taken away:

- Recognition works best when full sentences are used.
- Short phrases of two words have the worst recognition ratio. Therefore, it is better to avoid using them as voice commands.
- It takes an average of over five seconds to understand a command. This means that at the moment voice command cannot be used for critical commands that have to be understood immediately.
- With 76% success rate, voice recognition works quite well in general.

The tables 3 and 4 show the detailed results of the accuracy test.

	avg. time	**correct**	**words**	**Phrases**	**sentences**
User 1	10 s (3 s – 27 s)	69%	75%	50%	100%
User 2	6 s (4 s – 12 s)	100%	100%	100%	100%
User 3	6 s (3 s – 35 s)	93%	100%	75%	100%
User 4	8 s (3 s – 25 s)	81%	75%	75%	100%

User 5	4 s (3 s – 5 s)	100%	100%	100%	100%
User 6	9 s (4 s – 19 s)	44%	25%	50%	75%
User 7	11 s (4 s – 36 s)	44%	50%	0%	75%
Average	**7.70 s**	**76%**	**75%**	**64%**	**93%**

Table 3: Web Speech API accuracy test results per user

The following table is only taking correct answers into account. The times and number of tries of users who got no correct answer are not taken into the table.

	correct	**avg. time**	**avg. tries**
hello	6	4.37 s	1
play	3	4.13 s	1
how are you	6	3.97 s	1
go	4	4.18 s	1
new line	6	5.30 s	1.17
next	6	4.20 s	1
previous	4	6.93 s	1.5
is this working	6	5.79 s	1
help	6	7.18 s	1.33
I want to go home	7	4.43 s	1
stop now	5	5.79 s	1.4
delete	5	7.34 s	1.4
delete this	3	5.91 s	1.33
I need help	7	7.34 s	1
cancel	7	5.02 s	1.43
display error message	4	5.81 s	1
Average	**5.31**	**5.48 s**	**1.1**

Table 4: Web Speech API accuracy test results per phrase

The test has shown that the Web Speech API's `SpeechRecognition`, as currently implemented by Google Chrome, can already be used and has a reasonable recognition rate. Nevertheless, developers should take into account what types of commands work better and test the command they want to use. For this purpose, the accuracy test is available as an open source project on GitHub (Novy, 2013). It can be downloaded and changed to test all phrases that should be used in a project. The test has a theoretical support for different vendor prefixed versions

5.4 SpeechSynthesis API

In addition to the SpeechRecognition interface, the Web Speech API also specifies a way to control text-to-speech output. Currently, not even Google Chrome has any level of support for `SpeechSynthesis`. Support in at least Webkit (and subsequently Safari) can be expected in the foreseeable future, according to Chris Fleizach from Apple (Fleizach, 2013). Because there is currently no practical application of the SpeechSynthesis API, only a short overview will be given.

Like SpeechRecognition, SpeechSynthesis is completely JavaScript-based. The SpeechSynthesis object has three attributes: `pending`, `speaking` and `paused`. These attributes can be read only to get information about the current state of the speech output.

The following methods can be called from a `SpeechSynthesis` object:

`speak`
This method appends a `SpeechSynthesisUtterance` object to the end of the queue for spoken phrases. This means that the specified utterance will be spoken after all previously added texts have been processed.

`cancel`

This clears the queue and stops the current output, if an utterance is currently being spoken.

`pause / resume`

`pause` puts the `SpeechSynthesis` instance into the paused state and pauses utterances immediately, if something is currently being spoken. `resume` returns the `SpeechSynthesis` instance to the non-paused state and continues speaking.

`getVoices`

This function returns the voices that are available for this user agent.

SpeechSynthesisUtterance represent the different pieces of text that should be read. They have the attributes text (the text that should be spoken), lang, volume, rate (how fast the text should be spoken), pitch and voiceURI (which lets a developer specify the location of a speech synthesis service and voice that should be used).

There are various other details of the specification, like events, for the SpeechSynthesis API. For additional details, refer to the API specification (Shires & Wennborg, 2012)! In conclusion of this short overview, a simple example of how SpeechSynthesis could be used in the near future will be given:

```
var u = new SpeechSynthesisUtterance();
u.text = "This is a test.";
u.rate = 1.1;
u.volume = 0.5;
speechSynthesis.speak(u);

// It is also possible to use the constructor to add text
speechSynthesis.speak(SpeechSynthesisUtterance("This is a test."));
```

Listing 12: SpeechSynthesis example

5.5 Future browser support prospects

While, as mentioned before, at the time of the writing of this work only Google Chrome supports the Web Speech API, other browser vendors are planning to support it too. In the official MozillaWiki article “2013-Q1-Goals”, Speech API support is explicitly mentioned (“Platform/2013-Q1-Goals - MozillaWiki,” 2013). According to it, API implementation has already been finished, and the only thing missing is a backend service provider.

According to the WebKit bugtracker, implementation of the Speech API is planned for WebKit, too. While SpeechRecognition is not yet implemented, bug reports indicate that the work on Speech Synthesis is already quite advanced (“Bug 80261 – Implement Speech JavaScript API,” 2013; Fleizach, 2013).

6 CSS Speech Module

In addition to the beforementioned APIs, there is also a Speech Module for CSS3. It is currently specified as a Candidate Recommendation by the W3C (Weck, 2012). This module adds aural presentation information to HTML elements. This can be done in so-called Aural Cascading Stylesheets (ACSS), in the same way that visual styling information is added. ACSS attributes can simply be added like conventional CSS commands, like for example:

```
h1
{
   voice-family: paul;
   voice-stress: moderate;
   cue-before: url(audio/ping.wav);
   voice-volume: medium 6dB;
}
p
{
   voice-family: female;
   voice-balance: left;
   voice-pitch: high;
   voice-volume: -6dB;
}
```

Listing 13: A simple ACSS example

This would tell the browser to read a heading in a different voice and volume than a paragraph, as well as playing a specified sound cue before the actual heading is spoken. It is also possible to change the pitch, velocity, richness or angle in a 3D space as well as many other attributes.

Aural Stylesheets can be added to a website in the same way “normal” stylesheets are added. The only thing to remember is to set the `media` attribute to `aural`.

```
<link rel=”stylesheet” href=”stylesheet-aural.css” type=”text/css”
media=”aural” />
```

Listing 14: Inclusion of aural stylesheets

There are two main problems with ACSS. The first and gravest one is that as far as tests have shown no current browser has support for these properties. While testing of popular screen reader software like JAWS or Windows-Eye is beyond the scope of this work, there seems to be no level of support and no plans for future addition (Gibbins, 2010; Liebermann, 2007). According to Colin Liebermann, who claims to have spoken with the makers of JAWS, Freedom Scientific (Freedom Scientific, n.d.) back in 2007, it was arugmented that web developers probably couldn't apply aural stylesheets correctly, because they don't really understand what it is like to use soley a screen reader for interaction with a computer (Liebermann, 2007). The author of this work could find no traces that one of the major screen reader creators has since then added support for aural stylesheets.

Joe Clark also writes against the usage of aural stylesheets in his book Building Accessible Websites (Clark, 2002). He too argues that web developers simply lack the abilities to create aural stylesheets that work better than what dedicated screen readers are capable of:

> *"You simply don't have that training. Nor should anyone expect you to have it. Nor is there anywhere you can get that training." (Clark, 2002)*

While this book is by now over ten years old, which is certainly a long time in the web space, the pillars of his argumentation are still standing valid. Another point raised in the book is that ACSS would override carefully thought out and tweaked settings that screen readers might have defined over weeks and months of use with something that a web developer who might never have used a screen reader for more thant several minutes at a time found appropriate.

All in all, especially because of the lack of support, the additional effort needed for the creation of aural stylesheets would stand in no meaningful relationship to the benefit they might bring, even if the downsides mentioned before are completely dismissed.

7 Conclusion

There are many new or emerging web technologies that can be used to cater to people with physical or mental impairments. Many of them can already be used and work considerably well.

Concluding, the following paragraphs will give the answers to the central research questions mentioned at the beginning of this thesis.

Text Tracks work in most modern browsers and have a very good fallback-behaviour when support is missing, which means that developers can start using them now and don’t have to worry about older browsers.

Chapter 4 has shown that Speech Input works and is very straightforward to implement, but it is currently only supported by Google Chrome, and no other browser vendor has signaled any plans for future support. It is however already in use on various websites, including Google search, and because on browsers with no support input fields will still work properly, implementation can be considered without hesitation.

The Web Speech API is by far the most sophisticated API analysed in this thesis. As examined in chapter 5, it provides complex and powerful means for continuous speech recognition, which works considerably well at the moment. Like Speech Input, it will currently only work in Google Chrome, but many major browser vendors like Mozilla and Apple have announced that they are currently working on support.

Aural Stylesheets, as specified in the CSS Speech module, are currently not supported by any major web browser. In chapter 6 various problems with these stylesheets have been pointed out, especially the missing expertise of most web developers to include useful aural styling.

The following table shows an overview of current web browser support for the examined web technologies. In addition, the total percentage of users that can use an API are displayed, according to the statistical browser usage data from StatCounter from March 2013 (StatCounter, 2013). The fallback behaviour for web browsers without support is also depicted: Except for the Web Speech API, all technologies will not have any impact on web browsers without support.

Because the Web Speech API is not build on top of other technologies, it cannot fallback to anything and will simply not work.

	Text Track API	Speech Input API	Web Speech API	Aural CSS
Chrome 26	Basic	Yes	Yes	No
Chrome 28 Canary	Basic	Yes	Yes	No
Firefox 19	No	No	Planned	No
Firefox 21 Aurora	No	No	Planned	No
Internet Explorer 8	No	No	No	No
Internet Explorer 9	No	No	No	No
Internet Explorer 10	Multi-language	No	No	No
Opera 12.15	Basic	No	No	No
Safari 5 (Windows)	No	No	No	No
Safari 6 (Mac OS X)	Basic	No	Planned	No
Total support	**41.75%**	**37.23%**	**37.23%** **(+22.47% planned)**	**0%**
Fallback	**Yes**	**Yes**	**No**	**Yes**

Table 5: API browser support overview

In general, the author believes that it is important to know the limitations of these APIs in order to be able to work with them. If support shortcomings or quirks are known, then there is nothing to stop interested web developers to experiment with these techniques and use them in their projects.

7.1 Future work

There are many modern web technologies which have not been analysed in this paper simply because there are so many of them. This means that there are many possibilities to examine these technologies.

For the Text Track API, only subtiltes have been examined. The specification proposes other use cases too, which could also be further explored.

Another possibility for future work would be a more extensive speech recognition accuracy test. This could further help web developers to work with the Web Speech API and similar technologies.

In addition, the included browser support tests have to be repeatet every now and then to ensure that they are up to date.

References

Accessibility in Blended Learning. (2012). Retrieved from http://www.youtube.com/watch?v=OmW6NJow05Y&feature=youtube_gdata_player

Adobe. (n.d.). Adobe - Flash Accessibility Design Guidelines. Retrieved April 19, 2013, from http://www.adobe.com/accessibility/products/flash/best_practices.html

Apple. (n.d.). Apple – Was ist Safari – Ein besonders innovativer Webbrowser. Retrieved April 12, 2013, from http://www.apple.com/at/safari/what-is.html

Berjon, R., Faulkner, S., Navara, E. D., O'Connor, E., & Pfeiffer, S. (2013, March 24). HTML 5.1 Nightly. Retrieved March 24, 2013, from http://www.w3.org/html/wg/drafts/html/master/

Borodin, Y., Bigham, J. P., Dausch, G., & Ramakrishnan, I. V. (2010). More than meets the eye (p. 1). ACM Press. doi:10.1145/1805986.1806005

Bug 80261 – Implement Speech JavaScript API. (2013, March 21). Retrieved May 5, 2013, from https://bugs.webkit.org/show_bug.cgi?id=80261

Caldwell, B., Cooper, M., Reid, L. G., & Vanderheiden, G. (2008, November 12). Web Content Accessibility Guidelines (WCAG) 2.0. Retrieved from http://www.w3.org/TR/WCAG20/

Captionator. (n.d.). *GitHub*. Retrieved March 24, 2013, from https://github.com/cgiffard/Captionator

Chisholm, W., Vanderheiden, G., & Jacobs, I. (1999, May 5). Web Content Accessibility Guidelines 1.0. Retrieved April 25, 2013, from http://www.w3.org/TR/WCAG10/

Clark, J. (2002). *Building Accessible Websites*. Retrieved from http://joeclark.org/book/sashay/serialization/Chapter11.html#h2-1470

Estelle, J. (2011, April 27). Google Chrome Blog: Everybody's talking (and translating) with Chrome. Retrieved April 26, 2013, from http://chrome.blogspot.co.at/2011/04/everybodys-talking-and-translating-with.html

Faulkner, S. (2013, February 14). Using WAI-ARIA in HTML. Retrieved April 26, 2013, from http://www.w3.org/TR/2013/WD-aria-in-html-20130214/

Fleizach, C. (2013, March 7). Bug 106742 – Support WebSpeech - Speech Synthesis. Retrieved May 5, 2013, from https://bugs.webkit.org/show_bug.cgi?id=106742

Freedom Scientific. (n.d.). Products for low vision, blindness, and learning disabilities from Freedom Scientific. Retrieved April 28, 2013, from http://www.freedomscientific.com/

Gibbins, J. (2010, December 5). Aural CSS: Support for CSS 2 Aural Style Sheets / CSS 3 Speech Module | dotjay.co.uk. Retrieved April 28, 2013, from http://lab.dotjay.co.uk/notes/css/aural-speech/

Jacobs, I. (2005, October 14). 7 W3C Technical Report Development Process. Retrieved March 24, 2013, from http://www.w3.org/2005/10/Process-20051014/tr#maturity-levels

James, C., & Cooper, M. (2011, January 18). Accessible Rich Internet Applications (WAI-ARIA) 1.0. Retrieved March 24, 2013, from http://www.w3.org/TR/wai-aria/complete

Kloots, T. (2011, June 10). Easy Fixes to Common Accessibility Problems. *Yahoo! Accessibility Library*. Retrieved April 25, 2013, from http://yaccessibilityblog.com/library/easy-fixes-to-common-accessibility-problems.html

Liebermann, C. (2007, March 19). Why Screen Readers Don’t Support Aural CSS. Retrieved April 28, 2013, from http://www.cactusflower.org/why-screen-readers-dont-support-aural-css

Novy, F. (2013, May 3). SpeechAccuracyTest. *GitHub*. Retrieved May 5, 2013, from https://github.com/mydea/SpeechAccuracyTest

Pappas, L., Schwerdtfeger, R., & Cooper, M. (2010, August 16). WAI-ARIA 1.0 Primer. Retrieved March 24, 2013, from http://www.w3.org/TR/wai-aria-primer/

Pfeiffer, S. (2012, August 23). WebVTT support in browsers | Web Media Text Tracks Community Group. Retrieved March 24, 2013, from http://www.w3.org/community/texttracks/2012/08/23/webvtt-support-in-browsers/

Platform/2013-Q1-Goals - MozillaWiki. (2013, April 8). Retrieved May 5, 2013, from https://wiki.mozilla.org/Platform/2013-Q1-Goals#Media

Pultz, M. (2011, March 23). Accessing Google Speech API / Chrome 11 | mike pultz. Retrieved April 30, 2013, from http://mikepultz.com/2011/03/accessing-google-speech-api-chrome-11/

Sampath, S., & Bringert, B. (2010, October 18). HTML Speech Input API Specification. Retrieved April 26, 2013, from http://lists.w3.org/Archives/Public/public-xg-htmlspeech/2011Feb/att-0020/api-draft.html

Shires, G. (2013, February 21). Google Chrome Blog: Bringing voice recognition to the web. Retrieved April 26, 2013, from http://chrome.blogspot.co.at/2013/02/bringing-voice-recognition-to-web.html

Shires, G., & Wennborg, H. (2012, October 19). Web Speech API Specification. Retrieved April 26, 2013, from https://dvcs.w3.org/hg/speech-api/raw-file/tip/speechapi.html

Spector, D. (2011, November 3). Young People Think The Internet Is As Important As Breathing. *Business Insider*. Retrieved June 1, 2013, from http://www.businessinsider.com/internet-cisco-poll-2011-11

StatCounter. (2013, March). Top 12 Browser Versions (Partially Combined) from Jan to Mar 2013 | StatCounter Global Stats. *Top 12 Browser Versions (Partially Combined) from Jan to March 2013*. Retrieved April 12, 2013, from http://gs.statcounter.com/#browser_version_partially_combined-ww-monthly-201301-201303-bar

Thiessen, P. (2011). WAI-ARIA live regions and HTML5 (p. 1). ACM Press. doi:10.1145/1969289.1969324

W3C. (2013, June 1). All Documents (sorted by date) - W3C. Retrieved June 1, 2013, from http://www.w3.org/TR/tr-date-all

W3C. (n.d.). Ready to validate - Nu Markup Validation Service - W3C. Retrieved April 26, 2013, from http://validator.w3.org/nu/

W3Techs. (2013, April 25). Usage Statistics and Market Share of JavaScript Libraries for Websites, April 2013. Retrieved April 25, 2013, from http://w3techs.com/technologies/overview/javascript_library/all

Weck, D. (2012, March 20). CSS Speech Module. Retrieved April 26, 2013, from http://www.w3.org/TR/2012/CR-css3-speech-20120320/

List of figures

List of tables

List of listings

Attachments

A. CD

This CD contains all scripts that have been developed for this thesis as well as a backup of all cited sources, especially web-based sources.