Mobile TV as a Web Application

Mobile Web 2.0 - a new application framework for interactive Mobile TV

PER-ERIK SVENSSON

KTH Information and Communication Technology

Master of Science Thesis
Stockholm, Sweden 2007

COS/CCS 2007-21

Mobile TV as a Web Application

Mobile Web 2.0 - a new application framework for interactive Mobile TV

PER-ERIK SVENSSON

Academic supervisor and examiner:
Professor Gerald Q. Maguire Jr.

Company supervisor:
Stefan Andersson, Ericsson Research

Abstract

The existing advanced web browsers in today's mobile phones open up the door for mobile web applications. By using standard web technologies, a web page can be crafted to mimic the behavior of a normal application. The purpose of this master's thesis has been to look at web application development for mobile phones in general and to implement a web-based Mobile TV client to determine whether it would be a viable alternative to existing clients based on other technologies. The advantages are the same as for any other web application: 1) the user avoids the hassle of installing an application and will always run the latest version, 2) developers benefit from the browser's ability to render generic content, and 3) it is believed that the differences between browser implementations are less than in other environments in which an application would run, for example Java or operating system specific environments.

Sammanfattning

Mobil-TV som Webbapplikation

De avancerade webbläsare som redan idag finns tillgängliga på nyare mobiltelefoner har gjort det möjligt att skapa mobila webbapplikationer. Genom användandet av befintlig webbteknik kan en webbsida utformas på ett sådant sätt att den uppträder som en vanlig applikation. Syftet med det här examensarbetet har varit att titta på webbutveckling för mobiltelefoner i allmänhet samt att implementera en webbaserad MobilTV-klient för att avgöra hurivida en sådan lösning skulle kunna utgöra ett alternativ till redan existerande klienter baserade på andra teknologier. Fördelarna antas vara desamma som för andra webbapplikationer: 1) användaren slipper installera ett program och kommer alltid att köra den senaste versionen, 2) utvecklingen gynnas av webbläsarens förmåga att rendera godtyckligt innehåll och 3) skillnaderna mellan olika webbläsarimplementationer anses vara mindre i jämförelse med andra miljöer i vilka en applikation körs, till exempel Java- eller operativsystemspecifika.

Acknowledgements

First of all, I would like to thank Professor Gerald Q. Maguire Jr. for agreeing on being the examiner and academic supervisor of this thesis and for the valuable comments and corrections, carried out with the broad knowledge and patience that is to be expected from him.

Further, I would like to thank everyone at Ericsson giving me important feedback, most notably Stefan Andersson, Clinton Priddle, and Torbjörn Einarsson. Cristian Norlin provided me with the graphical user interface design and much of the graphics, for which I am grateful.

Contents

Chapter 1

Introduction

Today the Internet is synonymous with the Web for a lot of people. Applications that were previously downloaded or bought as hard-copies and installed natively, are being replaced with web-based alternatives with similar or better functionality. One of the best examples of this is perhaps web-based email clients, where many users depend solely on these services for their email and have been doing so for years. Some of these users have never used any other interface to access, manage and send email.

As web browsers have improved, the possibilities have vastly improved for web developers to create services that look and behave much like their desktop counterparts. A driving force for the service providers has been the advantage of having advertising-financed applications, as advertisements on the web are generally more accepted than intrusive adware programs.

Web browsers for mobile phones have just recently improved but are still lagging in functionality in comparison with desktop browsers, in part because of the limitations of the phones themselves. However, this is rapidly changing and there are already retail phones available that are equipped with browsers suitable for web application development.

This first part of this thesis examines the aspects of web development in general, focusing specifically on mobile phone browsers, with the specific goal of the implementation of a Mobile TV client in mind. The second part describes such an implementation, followed by an evaluation and comparison to Mobile TV clients based on other technologies.

1.1 Mobile TV

Mobile TV is believed to be the most successful 3G service with, as early as 2004, a 50% adoption rate in South Korea and an expected adoption rate of 10% for 2007 in Western Europe [1]. But, as noted by Carlsson and Walden, the success of Mobile TV will eventually depend on the content provided being attractive enough to retain a large user base [2].

The term Mobile TV is used to describe almost anything that involves the reception and playback of video content on a mobile device. The various technologies used can be categorized into three different groups. The first is unicast streaming, which is the only technology today that is commercially available on a large scale. Unicast streaming Mobile TV involves streaming of video content over existing cellular networks to handheld devices using a separate stream for each device.

Another approach to Mobile TV is that of using the existing radio network to stream content to several devices at the same time in a multicast fashion, maintaining a single stream for each channel. A technology for this is Multimedia Broadcast Multicast Service (MBMS), which will be available to the general public in 2008.

The third group consists of different, but similar, technologies for broadcasting Mobile TV using a separate radio network specially designed for TV broadcasting to portable devices. We will consider each of the technologies for Mobile TV used in cellular networks and broadcast networks in a section of its own.

1.1.1 Mobile TV in cellular networks

Unicast streaming

Traditionally, Mobile TV has been delivered via unicast streaming to a Real Time Streaming Protocol (RTSP) -capable player built in to the mobile phone. Usually, the links to the content are provided by the operator on a web (or WAP) portal. The next generation of Mobile TV for receiving unicast streaming consists of J2ME applications specially designed for this purpose. They usually offer added value in the form of Electronic Program Guides (EPGs) and interactivity such as voting, chatting, etcetera. The downside of this approach is that users need to download and install the program as well as any subsequent updates. Because of Java's security model the application also needs to be cryptographically signed and the user has to give his approval of the certificate and grant or deny the application access to the Internet.

MBMS

Multimedia Broadcast Multicast Service (MBMS) is an extension of existing third generation (3G) radio networks to be able to broadcast from a single point to multiple receivers via multicast. In contrast to technologies such as DVB-H and T-DMB, an uplink channel will be readily available. There are currently ongoing MBMS trials and the technology will be available to the general public as soon as operators have upgraded their base stations and MBMS capable handsets have been rolled out. This is expected to happen in 2008. MBMS is standardized within the 3rd Generation Partnership Project (3GPP).

1.1.2 Broadcast Mobile TV

DVB-H, T-DMB, and MediaFlo

There are three main competing methods of delivering Mobile TV by means of broadcasting over a separate radio network [3], in a similar fashion that ordinary digital TV is broadcast. In fact, the Digital Video Broadcasting, Handheld (DVB-H) standard is based on the commonly used DVB-T (T for Terrestrial) standard used to provide homes all over the world with digital television. DVB-H has been standardized by the European Telecommunications Standards Institute (ETSI) and it is being pushed by several companies, especially Nokia. Since it uses a separate radio network, operators need to acquire new licenses to operate in that spectrum or to partner with those who do have such licenses. In addition to telecom operators, many others are interested in broadcasting Mobile TV, which could lead to a battle for the few licenses that will be handed out. The regulations differ from country to country. In some countries it may not be possible for a telecommunications operator to acquire such a license.

The closest competitor to DVB-H is Terrestrial Digital Multimedia Broadcast (T-DMB) based on Digital Audio Broadcasting (DAB) radio and developed by a group of South Korean companies. The spectrum license regulations may be even harder for T-DMB, for example in the UK 80% of the DAB spectrum has to be used for audio broadcasting [4].

MediaFlo is a proprietary Mobile TV solution by Qualcomm, similar to DVB-H, but with a number of added features such as turbo coding and scalable video [3].

These technologies will of course require, besides a whole new broadcast infrastructure, handsets equipped with suitable Mobile TV receivers. As noted by Ollikainen and Peng, "Re-using existing DVB-T broadcast infrastructure for DVB-H is not enough to achieve the same coverage level as for DVB-T services" [5]. DVB-H handsets are already available from mobile phone manufacturers Nokia, Samsung and LG, among others.

Chapter 2

Mobile web development

This section gives a brief introduction to web development as well as outlines the prerequisites for mobile web application development.

2.1 HTML

When Tim Berners Lee created HTML (HyperText Markup Language), the main focus was on the hypertext functionality. HTML was inspired by, not based on, the Standardized Generalized Markup Language (SGML). The first application of HTML was to present abstracts of research papers [13]. From the beginning, HTML was intended to represent the logical structure of a text document. The physical representation of headings, paragraphs, etcetera was determined by the browser [6].

Ultimately, there were demands for more control over the visual appearance of web pages. Tags such as the `font` tag were introduced, allowing web developers to specify the typeface, size, weight, and color of text in HTML documents. Using table elements with zero-width borders, it was possible to do fairly advanced layout and positioning of page contents. Of course, the markup became cluttered and hard to comprehend. Today, HTML is once again written to represent the structure of documents, leaving all presentation related formatting to CSS (see next section).

In modern browsers, a HTML document is accessible through the Document Object Model (DOM) which is a World Wide Web Consortium (W3C) standard API for accessing elements in a document as objects. This makes it possible to use JavaScript to alter any element on the page. This is popularly known as dynamic HTML. A dynamic web application will require a certain level of DOM support.

2.2 Cascading Style Sheets

Cascading Style Sheets (CSS) is a stylistic language complementing HTML to provide styling and layout of page contents [7]. With CSS it is possible to define different styling for different media types. There are media types for rendering for a screen (assumed to be a screen of a desktop computer), handheld, printer, etcetera.

If the browser supports it, the media type `handheld` can be used to signal that a page has been *specifically designed* for a handheld device and thus preventing the browser from trying itself to adapt the page to the smaller screen. These techniques for adapting web pages to handheld devices, sometimes called Small-Screen Rendering or Smart-Fit, are essentially based on the browser replacing the stylesheet of a page with its own stylesheet with the intention of making a web page readable on a mobile device, despite the fact that the web page was not initially designed to be presented on a device with a small screen.

An essential feature of CSS when designing advanced user interfaces with a lot of graphical components is absolute positioning. It is very desirable to have full control of the placement of the various elements and also to be able to have overlapping elements. The effect that the small-screen rendering techniques usually have on pages that rely on absolute positioning is that the positioning is removed and the elements are floated below each other in the order they appear in the HTML document.

2.3 JavaScript

JavaScript was initially developed by Brendan Eich, of the Mozilla Foundation, who at that time worked for Netscape. JavaScript has little to do with Java besides some similarities in syntax, the name was just a marketing ploy by Netscape and Sun Microsystems. The name was changed from LiveScript to JavaScript at the last minute. Today JavaScript is standardized by ECMA (European Computer Manufacturers Association) and so its official name is ECMAScript. Technically, JavaScript refers to Mozilla's (and Netscape's) language implementation of ECMAScript, although most people simply call the language JavaScript [8].

JavaScript is most commonly used for client-side scripting, i.e., running code inside the web browser. Once a web page is downloaded, JavaScript code can dynamically alter the page and respond to events triggered by the user. By using JavaScript to alter a web page loaded in the browser, mimicing a normal application, the web page turns into a thin-client that is loaded from the server on request and runs locally in the web browser.

To be able to provide a dynamic user interface that behaves just like an application, we can utilize JavaScript. Additionally, JavaScript let us communicate with the server without having to reload the page.

A web page can also invoke operations on the server, previously called Remote Scripting, but now often called Ajax (Asynchronous JavaScript and XML), a broad term covering the various techniques for asynchronous client-server communication within web pages. The most common method used today is `XMLHttpRequest`, an API for making such requests with JavaScript. Alternatively, one can use HTML elements like `iframe`, `script`, and `img` and set their respective `src` (source) properties to have them update dynamically, thus making a request to the server.

See [9] for detailed descriptions of the various methods that can be used for remote scripting.

Despite the name `XMLHttpRequest` it is not necessary to use XML as the data exchange format. One can equally well use plain text, HTML, JSON (see section 2.4.2) or any other format of choice. (Binary data may have to be BASE64-encoded though). In fact, there are several large Ajax sites that do not use XML at all.

2.4 Data exchange formats

When an asynchronous request for data is made, the server needs to represent the data in a way that can easily be accessed from within JavaScript once it has been transferred to the client. Besides proprietary text formats, there are two very commonly used data exchange formats: XML and JSON.

2.4.1 XML

XML, short for eXtensible Markup Language, is a generic markup language for data representation. XML was derived from SGML. Anyone can create their own document with unique tags and attributes. XML supports namespaces, thus one can mix different kinds of XML documents together into a single document. XML is the most common data exchange format used in web applications today, hence the 'x' in Ajax.

If the browser has built-in support for XML, this format can be utilized for the asynchronous client-server communication. When `XMLHttpRequest` receives an XML response, a DOM tree is automatically built up from the XML and standard DOM functions can be used to access the data. A DOM tree is the resulting data structure when building up the hierarchical level of elements in an XML document as objects. When the data to be transferred is already in XML, such as an SVG or an XHTML document, it seems logical to keep this format rather than transform the document to some other format.

2.4.2 JSON

JavaScript Object Notation (JSON) is a subset of JavaScript and described as "the fat-free alternative to XML" [10]. JSON is heavily used in many Ajax applications on the web today. Because JSON is essentially JavaScript, it is very easy to handle such a response within the browser. Providing that one can *trust the data not to contain any malicious JavaScript code*, one can simply run eval() on the string containing the data to have it turned into an object. Alternatively, there are very compact JavaScript libraries available for safely parsing JSON data. The simplicity makes it attractive to use JSON in a web application, especially when targeting a browser without XML support. There are numerous libraries and built-in JSON support for many different programming languages. JSON is defined in RFC 4627[11].

2.5 SVG

Scalable Vector Graphics (SVG) is a W3C recommendation for using XML to describe graphics as lines, curves, and text [12]. SVG can also be used to create animated graphics. What really makes SVG suitable for web applications is that it can work in conjunction with (X)HTML and JavaScript [13]. The bad news is that the current subset profile of SVG for mobile devices, SVG Tiny version 1.1, does not support scripting. While the W3C candidate recommendation SVG Tiny 1.2 supports scripting there are currently no implementations available. Additionally, SVG is often not implemented natively in the browser, but instead as a browser plugin, which may complicate the integration with the rest of the web page.

SVG has its own Document Object Model, the SVG DOM, compatible with the original DOM, but with added features specific to SVG. SVG Tiny 1.2 uses a subset of the SVG DOM called the SVG Micro DOM (uDOM). It is expected that mobile browsers that will support SVG Tiny 1.2, also will support the SVG uDOM.

Even if SVG might not be used to render a user interface on a mobile device, it may still be used for graphics content, since it allows scaling. There are several drawing programs available, for example Inkscape [1], that can generate SVG documents and libraries that can be used to rasterize the resulting graphics into ordinary bitmap images.

2.6 Flash

Flash is a platform for creating rich multimedia applications. With the help of a plugin, these applications can run inside the browser, seemingly integrated in a web page. Flash is a proprietary format of Adobe Systems Inc.

Flash Lite is a mobile profile of Flash. Flash Lite 1.0 and 1.1 are based on Flash 4 that was released in 1999. Flash Lite 2.0 is based on Flash 7. The most common version of Flash Lite shipped with mobile phones is 1.1 as of this writing. Flash Lite 1.1 has quite a few limitations compared to Flash Lite 2.0, which in turn has several limitations compared to the desktop version of Flash. Most notably, there is no support for streaming video in Flash Lite 1.1. Flash Lite 2 can support streaming video, given that the device inherently supports streaming video. For streaming to work, some specific conditions have to be met. For example, it is known to work on a Nokia 6620 with Flash Lite 2 installed, streaming a file encoded with Helix Mobile Producer from a Helix Streaming Server over the T-Mobile carrier, with an unlimited data plan [14].

2.7 XHTML

The eXtensible HyperText Markup Language (XHTML) is a W3C initiative to make HTML more like XML, bringing a more strict syntax and parsing to HTML. As

[1]http://www.inkscape.org/

mentioned earlier, one of the benefits of treating HTML as XML is that one can use namespaces to mix XHTML with other XML-based documents. Specifically, one can merge an SVG document into an XHTML document and have them both being part of the same DOM tree, to build a true graphical web application. Provided that the browser supports this, of course.

2.8 HTML5

HTML5 [15] is an initiative by the Web Hypertext Application Technology Working Group (WHATWG), an unofficial group formed by Apple, Mozilla, and Opera with the aim of making it easier to develop web applications. The most famous addition to HTML in HTML5 is probably the `canvas` tag, implemented in Apple's Safari and Mozilla's Firefox browsers. The `canvas` tag allows an arbitrary graphics canvas to be placed on a web page. One can then draw on the canvas with JavaScript calls.

2.9 WAP

When writing about web development for mobile devices, it is hard not to mention the Wireless Application Protocol (WAP). The first version of WAP uses its own markup language, the Wireless Markup Language (WML), for pages especially designed for mobile devices. WAP 2.0 uses a mobile profile version of XHTML, XHTML-MP, as its markup language.

When WAP was first introduced, terminals had very limited screens (small, non-color) and used circuit switched data links that were both slow and took a long time to set up. People willing to embrace the new technology were often turned off by the bad user experience resulting from this. The hype surrounding WAP nourished the disappointment. The reputation of the mobile web still suffers from the early failures of WAP.

Chapter 3

Mobile web browsers

This section describes the most common advanced mobile web browsers, as well as some initial findings resulting from testing and experimenting with a couple of mobile phones. We consider an advanced mobile web browser a browser with similar capabilities as a standard desktop browser.

3.1 Netfront

Netfront is a browser targeted specifically towards mobile devices [17]. It is being developed by a Japanese company called Access. Since this browser ships with several Sony Ericsson phones, and also many Samsung handsets, it will be given a lot of attention in this thesis. Netfront on Sony Ericsson phones is the only browser tested that has a *stable* streaming video plug-in, which is a requirement for our Mobile TV implementation.

Testing of the Netfront browser has been carried out mainly on a Sony Ericsson K800i with the latest official software as of this writing, revision R1GB001. This phone is running Netfront 3.3. Additionally, some tests were made on a HTC Smartphone running Windows Mobile 5 and the Netfront 3.4 Technical Previews 1-3 of the upcoming version 3.4. Unless otherwise noted, conclusions refer to testing on the K800i.

The adaptation of ordinary web pages to the small screen of mobile devices is called Smart-Fit Rendering in Netfront. Unfortunately, there *seems to be no way of turning this off or even detecting that Smart-Fit is on.* CSS media types `screen` and `handheld` are both applied regardless of setting. In Netfront 3.4 TP2, only the `screen` media type is applied, with or without Smart-Fit. It is up to the user to change this setting in the options menu of the browser. With Smart-Fit turned on, all CSS absolute positioning is removed and all elements are placed below each other in the order they occur in the markup. Therefore, one should see to it that the first rendered element will be a box, covering the visible area of the page before scrolling, containing a notification to the user to turn the Smart-Fit setting off. When Smart-Fit is turned off, this element is hidden beneath the elements of the user interface and/or the video plugin. After a while, when the user supposedly has read the notification and turned Smart-Fit off, the box can be removed completely from the page with a JavaScript call.

The SVG support which has been implemented is sufficient enough that one would consider using it for much of the graphics of the user interface. According to the specifications of Netfront 3.3, the SVG level support is SVG Tiny 1.1. Although it is not possible to have the entire UI as a single SVG, since scripting of SVGs is not supported, the user interface could in theory be built up as a set of different SVGs much in the same way as one would do with bitmaps. It has been verified that one can assign a link to an SVG element and have it yield a JavaScript event whenever it is focused or clicked on. There are at least two things that make this solution less attractive. First, transparent areas of an SVG are not transparent with respect to the entire page. For example, a non-rectangular shape will appear in the page with its bounding box background as a white surface. Secondly, SVGs are always rendered in front of other elements so one can not have an SVG as background and then place HTML text on top of it, regardless of any `z-position` attributes. To sum it up, you can't use SVG as a foreground **or** as background. Also, a gradient in a basic SVG image created with Inkscape was not rendered at all, simply because SVG Tiny 1.1 does not support gradients [18]. Version 3.4 of Netfront will supposedly support SVG Tiny 1.2 with uDOM, but 3.4 TP2 has no support for SVG at all.

Three methods can be used for remote scripting in Netfront 3.3. The first and perhaps the easiest to implement is to update the `src` (source) attribute of a `script` element, in which the browser downloads the content from the specified location and, since it's a `script` tag, immediately tries to evaluate the new content as JavaScript. Note that an `onload` event is **not** generated when the content has finished loading, so if a variable is assigned a value, a callback function has to be executed at the end to signal that the data is available. As an alternative, data can instead be passed as an argument to the callback function, which will not be called until the data is loaded.

Another method is to have a hidden `iframe` and set the `location` property whenever an asynchronous request is to be made. A load event is triggered when the iframe has finished loading, so that one knows when the content is ready to be processed.

The third method for remote scripting in Netfront 3.3 is to set the `src` attribute of an image and have the browser download a single pixel image and at the same time have the server set a cookie. An `onload` event occurs when the image has loaded (and the cookie is set), and then the cookie contents can be extracted. According to RFC 2109 [16], browsers should allow cookies to be as large as 4096 bytes and allow for at least 20 cookies per unique host or domain. Whether Netfront obeys this has not been tested.

In both cases, an obtrusive and rather annoying progress bar is displayed during the request. While this could perhaps be acceptable for user generated events, such as changing channels, responding to interactivity events, etcetera, it is highly undesirable to have the progress bar show up on timed events, for example. Netfront 3.4 introduces `XMLHttpRequest` and such requests will probably not make the progress bar appear, as it is designed for background loading and no known browser uses a process indicator for these requests. When tested with the Windows Mobile version of Netfront 3.4, the progress indicator did not indicate progress on XMLHttpRequests.

When dealing with user generated JavaScript events in Netfront on the K800i, it is very unfortunate that the joystick does not yield keypress events when moved, only a click on the joystick generates a keypress event. The only option is to let the user navigate between links and use the focus event on the links to determine in what direction the user is moving, which will be discussed further in section 4.3. On the HTC Smartphone running Windows Mobile and Netfront 3.4 TP2, key events are generated when moving the joystick as well as when toggling the scroll button on the side of the device. However, keydown events are incorrectly repeated when pressing and holding a button on that handset.

One of the most annoying bugs in Netfront 3.3 it that when the `overflow` property of a `div` element is set, the text overflowing the `div` box is correctly hidden, but if the hidden text stretches outside the visible screen area to the right, then scrollbars appear. Note that this only applies to the right side of the screen, when overflowing at the bottom of the screen everything works as intended and expected.

It would be nice to have the clear key 'C' generating an event, so that it could be used as backspace if one would like to handle user input with JavaScript instead of the browser's built-in textual input. Unfortunately, keyup events do not work as intended. A keyup event is always issued within half a second, even if the user holds the button for a longer period of time. If one would like to emulate the built-in textual input, pressing and holding a button should yield the corresponding digit.

There are different view settings for normal, full screen, and horizontal that can be set by the user. When the view changes the `innerWidth` and `innerHeight` properties of the `window` object are properly updated, but a `resize` event is not generated to let the application know that the visible screen area has been updated, so that it can update the user interface accordingly. This was reported to Access in early March 2007, a couple of weeks before the release off TP3, but it was not fixed in that technical preview version. The most recent technical preview version, as of this writing, (TP4) fails to start on the HTC MTeoR handset.

Access is currently working on a new version of Netfront, version 3.5. Highlights in this version will be something they call "Embedded Ajax", non-standard JavaScript APIs for improving web applications for mobile devices. For example, there will be a call for doing explicit garbage collection. The release date for Netfront 3.5 is unknown.

3.2 Opera Mobile

The Norwegian software company Opera has been in the web browser market for many years. The mobile version of their desktop browser, simply called "Opera Mobile", comes pre-installed on Sony Ericsson's Symbian models, Nokia's Communicator series, and many Motorola handsets. Moreover, the browser can be downloaded and installed on a wide range of handsets [19].

Opera uses their Small-Screen Rendering to adapt web pages to the small screen of the mobile device. Fortunately, by setting the CSS media type to `handheld`, one can prevent unwanted rearranging of pages specifically designed for a mobile browser.

The Opera Mobile browser version 8.60 pre-installed on a Sony Ericsson W950 with Symbian UIQ was used for testing. The latest version of Opera Mobile is 8.65 as of this writing. Highlights in the newest version include, besides a slimmer title bar freeing up valuable screen area, two-button browser shortcuts. Shortcut keys are functions in the web browser mapped to certain keypad keys to provide easy access. To use a shortcut key in version 8.65, the user needs to press # before the shortcut key. Thus, ordinary key presses can be used by web applications without the user having to change the brower's settings as in previous versions and in the Netfront browser.

SVG support is implemented as a plugin and although it can render some complex SVG images, it suffers from the same problems as the Netfront implementation with regards to transparency and z-positioning. Also, it had the exact same problem with the gradient as with Netfront, indicating that the level of SVG support is SVG Tiny 1.1.

The version of Opera Mobile on Sony Ericsson W950 is said to include `XMLHttpRequest` and this has been verified to be true. Other methods of doing remote scripting have not been tested. It is generally preferred to use XMLHttpRequest when available, since it is a defined API specifically for making asynchronous requests to the server and other methods are just tweaks to achieve the same thing. In Mozilla Firefox on the desktop, XMLHttpRequest was significantly faster compared to the hidden `iframe` technique.

Key related events work as they should, even the power button on top of the phone as well as the play/stop and volume buttons on the side, yields a keypress event. However, scrolling the scroll wheel up and down gives a key code of zero in both directions so it is difficult to know in which direction the wheel was turned. Unlike Netfront, Opera Mobile supports the resize event when the user changes the view settings.

Opera is about to release version 9 of its mobile browser. New in this version is, among other things, improved SVG support. The release date for Opera Mobile 9 is yet to be announced.

3.3 Nokia web browser

A couple of years ago, Nokia decided to develop their own mobile web browser based on Apple's WebKit [20] upon which Apple's Safari [21] browser is based. WebKit, which is open source, was originally derived from the open source components used in the Konqueror browser that ships with the desktop environment KDE [22].

This browser was tested on a Nokia N73, software version 2.0628.0.0.1 and on a Nokia N95 with an unknown software version, most likely the version shipped with the first of these phones ever sold.

The initial tests show that on the Nokia N73, there is no support for SVG in the browser. SVG documents are opened in a separate viewer. Unfortunately, the level of SVG support on the N95 was not checked during the limited time the phone was available for testing. Surprisingly, there seems to be no support for PNG images on the N73, which one would consider basic functionality. Nokia N95 has support for PNG images, even with transparency.

The tested Nokia web browser on the N73 and N95 includes `XMLHttpRequest`. Other techniques for remote scripting have not been tested, but are presumably available. When the browser supports `XMLHttpRequest`, support for other techniques becomes irrelevant.

Key events are generated for most keys and since browser shortcuts are always on and cannot be turned off by the user, one has to call the preventDefault() method of the event object to override the shortcuts and prevent them from being activated. Some default actions can not be prevented though. The joystick steers a mouse pointer, so it does not trigger key events. Instead it generates mouse events, at least on the N95 where it was tested for. It is likely that the joystick generates mouse events on the N73 as well.

The Nokia web browser supports the resize event when the user changes the view settings.

3.4 Others

3.4.1 Minimo

Minimo [23], short for Mini Mozilla, is a project of the Mozilla Foundation to create a mobile browser based on the Gecko rendering engine [24]. A single developer has been assigned to the task during the last couple of years, but the project is now currently on hold. Minimo does not come pre-installed with any known device so it has to be downloaded and installed separately. When tested on the Windows Mobile HTC MTeoR, Minimo runs very slow, but the rendering is excellent, as expected from the Gecko engine. Minimo comes with a smallscreen.css file that is used by default. It can be disabled by user settings. The hefty CPU and RAM requirements and the uncertain future of the project, makes Minimo less attractive than the browsers previously described.

3.4.2 Pocket Internet Explorer

Pocket Internet Explorer (PIE) is Microsoft's browser for mobile devices. It is **not** based on the desktop browser Internet Explorer. The version of PIE installed on the Windows Mobile HTC MTeoR smartphone seems to have only limited support for JavaScript. CSS support is also limited, specifically there seems to be no support for inline styles. By adding a proprietary meta tag `MobileOptimized` to a page, the fit-to-screen rendering of Pocket Internet Explorer can be turned off. It is apparent that the PIE implementors are aiming for IE compliance rather than standards compliance; hence if one were to target the PIE browser one would have to make all the special exceptions in the code to handle the Internet Explorer specifics. As a side note, when the phone is delivered, one of the Favorites (the IE name for Bookmark) links to the download page of Netfront for Windows Mobile.

Chapter 4

Implementation

4.1 Background

The goal has been to develop a Mobile TV client based on web technologies that *includes all the important features in existing clients* based on other technologies. The work was carried out at Ericsson Research, where the client specification was set, including all the features that were to be implemented. While watching live TV at full screen width one can browse a list of available channels with channel icons, and names and titles of the currently running programs. Additionally, if Electronic Program Guide (EPG) data is available for the current channel, one can read additional information about the program being watched and the upcoming programs on that channel.

If the keypad key hash (#) is pressed when watching a TV channel, the EPG enters full-screen mode with the video element as a thumbnail in the upper left corner along with information about the program being watched. In full-screen EPG mode, the user can browse program information for all channels that have this information available and also be able to read it without having to scroll.

The client has support for fully dynamic interactivity events that can appear on a per channel basis at a specified time. Typical such events can be for example voting, or banners with or without the possibility to purchase goods directly. Interactivity will be discussed further in section 4.5.

Whenever the star key (*) is pressed the client poses a service selection where the user can currently choose either the live TV service or Video on Demand (VoD). When entering VoD the user is given a list of video clips to select from. The list consists of a thumbnail icon and the title of the clip. By clicking on an entry the video clip starts playing with the video shown in full screen width. Along with the thumbnail icon and the title of the current clip, a number of stars are shown, indicating the rating of the clip as determined by a poll. When the clip has finished playing the client returns to the list of available clips. The VoD functionality is very basic because it has been implemented just as a proof-of-concept.

The server that provides the streaming TV channels along with the EPG data, is an existing in-house implementation in Python using the event-driven networking engine Twisted [25]. Channel switching is achieved by switching the Real-time Transport Protocol (RTP) stream inside an established RTSP session. The client requests a change of channel by sending a command over HTTP. Only minor adjustments were required on the server side as described in section 4.4; where we go into details about the client-server communication.

One of the requested features of the Mobile TV web application has been to allow for easy customization of the graphical user interface (GUI). This has resulted in a solution where a user interface designer can essentially draw the GUI as a whole and have it compiled and merged with the program logic to produce a client with the mere press of a button. Combined with the thin-client nature of a web application as previously discussed in section 2.3, this adds the possibility to have skins or themes that could be updated on a day-to-day basis.

The web-based Mobile TV client was developed based on the Model-View-Controller (MVC) software architectural paradigm [26]. MVC involves splitting the logical and the presentational, as well as the user interactive parts, into separate layers. The reason for doing this is that one layer can updated without affecting the others, which will ease development in the long run. In this implementation, the View represents the HTML document (with images) comprising the GUI, the Controller represents the JavaScript object handling user input, and the Model represents the actual content retrieved from the server.

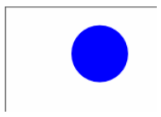
4.2 Graphical User Interface

As previously noted, SVG is a suitable format for representing vector graphics and it would therefore be attractive to use it for the GUI. However, it has been concluded, based on the observations in the previous chapter, that SVG support on mobile phones is not yet sufficient enough. Could there perhaps be a way for us to still use SVG when creating and storing the graphical user interface, but not as a part of the actual client? The answer: yes, there is.

There are important reasons for wanting to use SVG initially. First of all it is much easier to draw and update vector graphics than when working with bitmap images. Secondly, since it is assumed that we will be dealing with different screen sizes, we can easily scale the GUI to any size, preferably keeping the same aspect ratio. Another important fact is that if we choose to use SVG from the very beginning, the same graphics could be reused once SVG is commonly available in the mobile web browsers.

With this as the starting point, how can we begin by defining a set of SVGs representing the GUI, in order to produce graphics to be used in a browser with little or no SVG support. For this specific purpose, a program for transforming SVGs has been written in Java, using the Batik SVG Toolkit [27]. Batik is an open source library mainly for manipulating SVG documents, although it includes general XML parsers and functions for working with DOM trees as well. It is licensed under the Apache license version 2.0, which is an Open Source Initiative (OSI) approved license.

When working with SVG documents, an element or a group of elements can have an identification (`id`) attribute assigned to it/them. The reason for having this is usually that one wants to be able to address a specific element in the SVG from a script. After identifying it, we can operate on that element, move it, change its color, make it visible/invisible, or do any other operation that is supported by the element. This is an essential part of the Document Object Model (DOM) discussed earlier. Since HTML is also represented as a DOM structure, we could do the same with an element of an HTML document. Naturally, HTML has less features than SVG, so some operations would not be available, for example, if we were to alter the attributes of a bitmap image element containing a circle, rather than altering the attributes of a circle element in SVG.



```
<svg>
    <circle cx="100" cy="50" r="30" style="fill: blue"/>
</svg>

<html>
    <div style="position: absolute; left: 70px; top: 20px;">
        <img src="bluecircle.png">
    </div>
</html>
```

Figure 4.1. A filled blue circle at a specific position and the SVG and HTML markup to render it.

We start by considering an SVG definition of a part of a graphical user interface, then think about how it could be represented as a web page. What happens when we interact with a GUI is typically that various objects appear or disappear, they move around, and sometimes they may even change color or shape. In order to achieve this on a web page, each object that is dynamic has to be contained in an element of its own. For graphics, this means that every dynamic object has to be a separate bitmap image. Every piece of dynamic text will need its own container too.

The SVG transformer program mentioned above has been developed for the sole purpose of turning SVGs defining a GUI into a web page capable of expressing all the states of the same GUI, using HTML with bitmap images and text. For every element in the SVG that will be a dynamic component of the user interface, a specific `id` is set, e.g. `selector` or `icon`. The relevant ids are specified in a separate

XML file with additional information used in the transformation. The transformer program renders the SVGs internally. The bounding box of an element defines the area that will be rasterized into the corresponding bitmap image and the top left corner of that bounding box determines where that image will positioned in the resulting web page. When an image is extracted, all other elements in the SVG are hidden so that they will not appear in the image if there are overlapping elements. All objects, text and images, are then put in an HTML document and wrapped into `div` elements that are absolutely positioned with CSS. The ids are carried over from the SVG elements to the corresponding HTML `div` with a specific view name (see below) prepended to make sure that the `id` will be unique within the HTML document.

What about shaping or changing the color of an object then? We can not reshape a bitmap image or change its color by means of scripting. What if the user is pressing a green button and it is supposed to turn red? The good news is that we do not have to change the color, instead we create one image for each color and select which one is visible at a given time. The same technique is used to create the appearance of an object changing shape. This approach is not novel, it is the same basic principle used in early animation to create the notion of motion. When implementing the client it was only once necessary to use this method and that was when making the appearance of unlightened stars when indicating how many stars that were given to a specific video on demand clip as a result of voting.

The graphical user interface was designed by usability researcher Cristian Norlin, with Ericsson Research, who also made the graphics for all the components comprising the GUI. The design is based on *views*, where a view is a distinct part of the GUI in which the user is interacting at a given point in time and from where he can navigate to other views. A typical view is shown in figure 4.2 (See appendix A for the complete set of views). Each view is defined in its own SVG document, but when compiled into HTML, all views reside in the same document with a separate `div` element holding each view. The concept of views has been adopted in the client logic as well, where each view is represented by its own object. Thus, switching between views can be accomplished visually by hiding the current view and unhiding the new view and logically by updating a reference variable to point to the now current view. This is similar to the concept of *cards* in WML, where a single markup document holds several *cards*, but only one card is visible at a time [28]. The benefits of using this approach will be made clear in the next section when we look at how to handle input control.

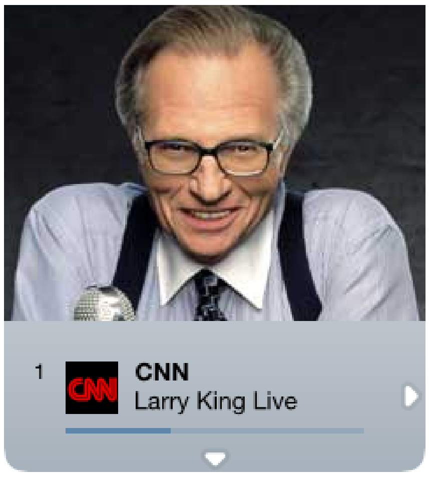


Figure 4.2. View A1. This is the main view of the Live TV service. It shows some information about the channel being watched. Navigate right to the A2 view for a list of programs on the current channel. By navigating up or down, you will be taken to the B1 view, which is a list of available channels.

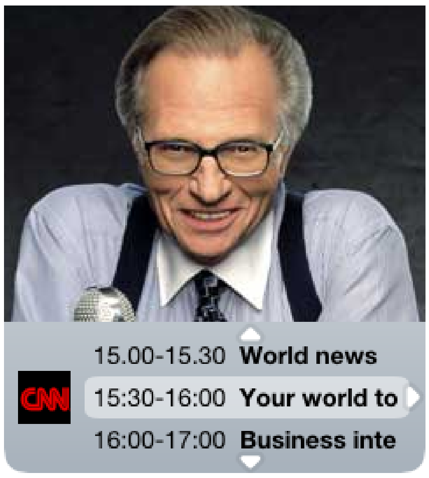


Figure 4.3. View A2. This view shows a list with the current and the nine upcoming programs on the channel currently being watched. Browse the list by navigating up or down, and navigate right to go to the A3 view, which will show you all information about the selected program. Navigating left will take you back to the A1 view.

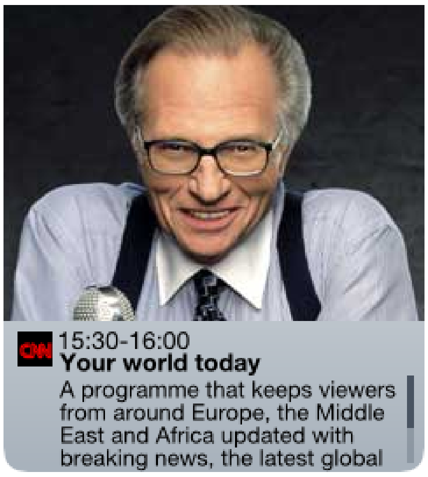

15:30-16:00
Your world today
A programme that keeps viewers from around Europe, the Middle East and Africa updated with breaking news, the latest global

Figure 4.4. View A3. This view shows information about a program on the currently watched channel. Navigate left to go back to the A2 view or right to go back the A1 view. The program description text is scrolled by navigating up or down.

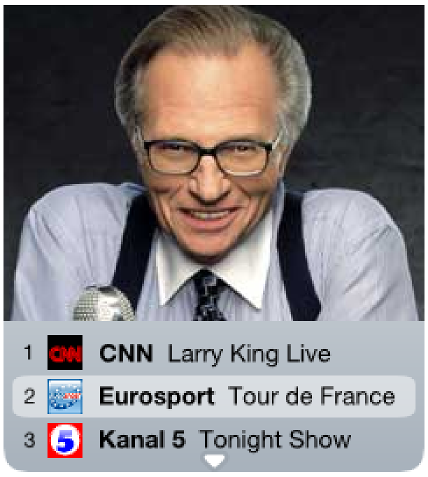


Figure 4.5. View B1. This view shows a list of all available channels. Browse the list by navigating up or down. By clicking on an item, a channel change is initiated and you will be taken to the A1 view. Navigate left to go back to the A1 view without changing channel. Being idle for ten seconds will also bring you back to the A1 view.

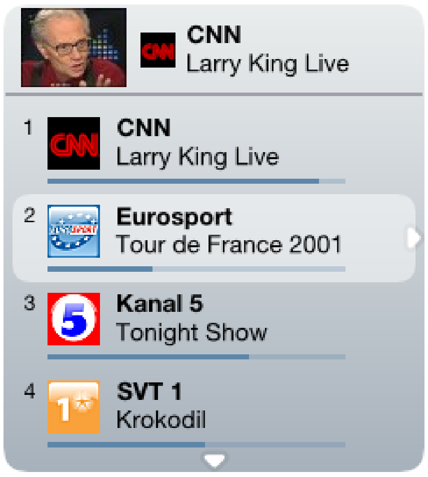


Figure 4.6. View D1. This is the full screen equivalent of the B1 view. You enter this view by pressing the hash key in the A1 or B1 view or by navigating here from the D2 view. Browse the list by navigating up or down. By clicking on an item, a channel change is initiated and you will be taken to the A1 view. Navigate left or press the hash key to go back to the A1 view without changing channel.

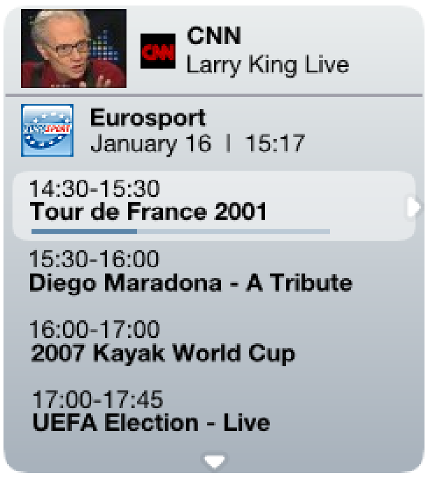
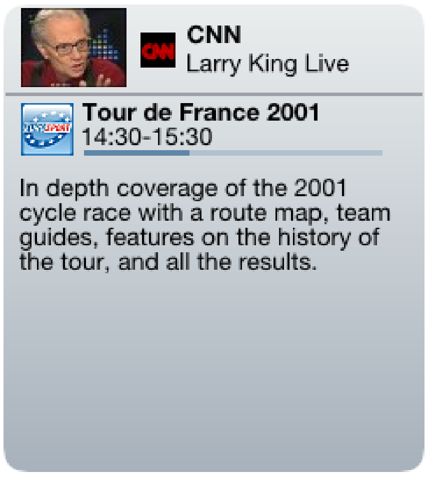


Figure 4.7. View D2. This is the full screen equivalent of the A2 view. You enter this view by pressing the hash key in the A2 view or by navigating here from the D1 or D3 view. Browse the list by navigating up or down, and navigate right to go to the D3 view, which will show you all information about the selected program. Navigating left will take you back to the D1 view. Press the hash key to go back to the A1 view.



CNN
Larry King Live
Tour de France 2001
14:30-15:30

In depth coverage of the 2001 cycle race with a route map, team guides, features on the history of the tour, and all the results.

Figure 4.8. View D3. This is the full screen equivalent of the A3 view. You enter this view by pressing the hash key in the A3 view or by navigating here from the D2 view. If the program description text is too long, a scrollbar will appear, which you can scroll by navigating up or down. Navigate left to go back to the D2 view or press the hash key to go to the A1 view.

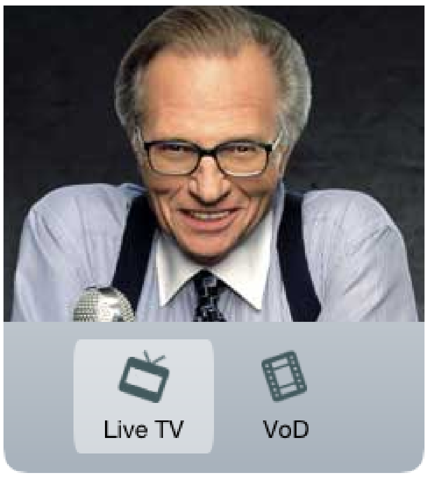


Figure 4.9. View C1. This is the service selection view. Pressing the star key at any time brings up this view. Select an item by navigating left or right and click on it to start the service. If you select the Live TV service you will be taken to the A1 view, and the V1 view for the Video on Demand service. Press the star key again to discard this view without taking action.

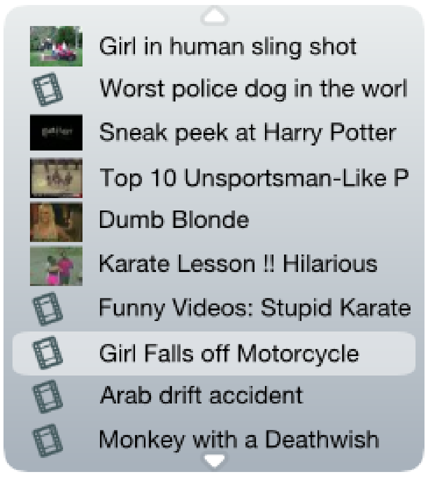

Girl in human sling shot
Worst police dog in the worl
Sneak peek at Harry Potter
Top 10 Unsportsman-Like P
Dumb Blonde
Karate Lesson !! Hilarious
Funny Videos: Stupid Karate
Girl Falls off Motorcycle
Arab drift accident
Monkey with a Deathwish

Figure 4.10. View V1. This view shows a list of video on demand clips. You enter this view by selecting "VoD" in the C1 view or if you are returning from the V2 view. Browse the list by navigating up or down. By clicking on an item, you will be taken to the V2 view where the clip will start playing, if it is not already playing. All clips that have been previously viewed will show up with a thumbnail, if a thumbnail is available.

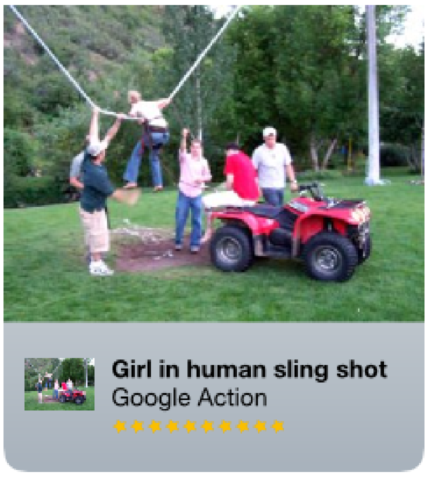
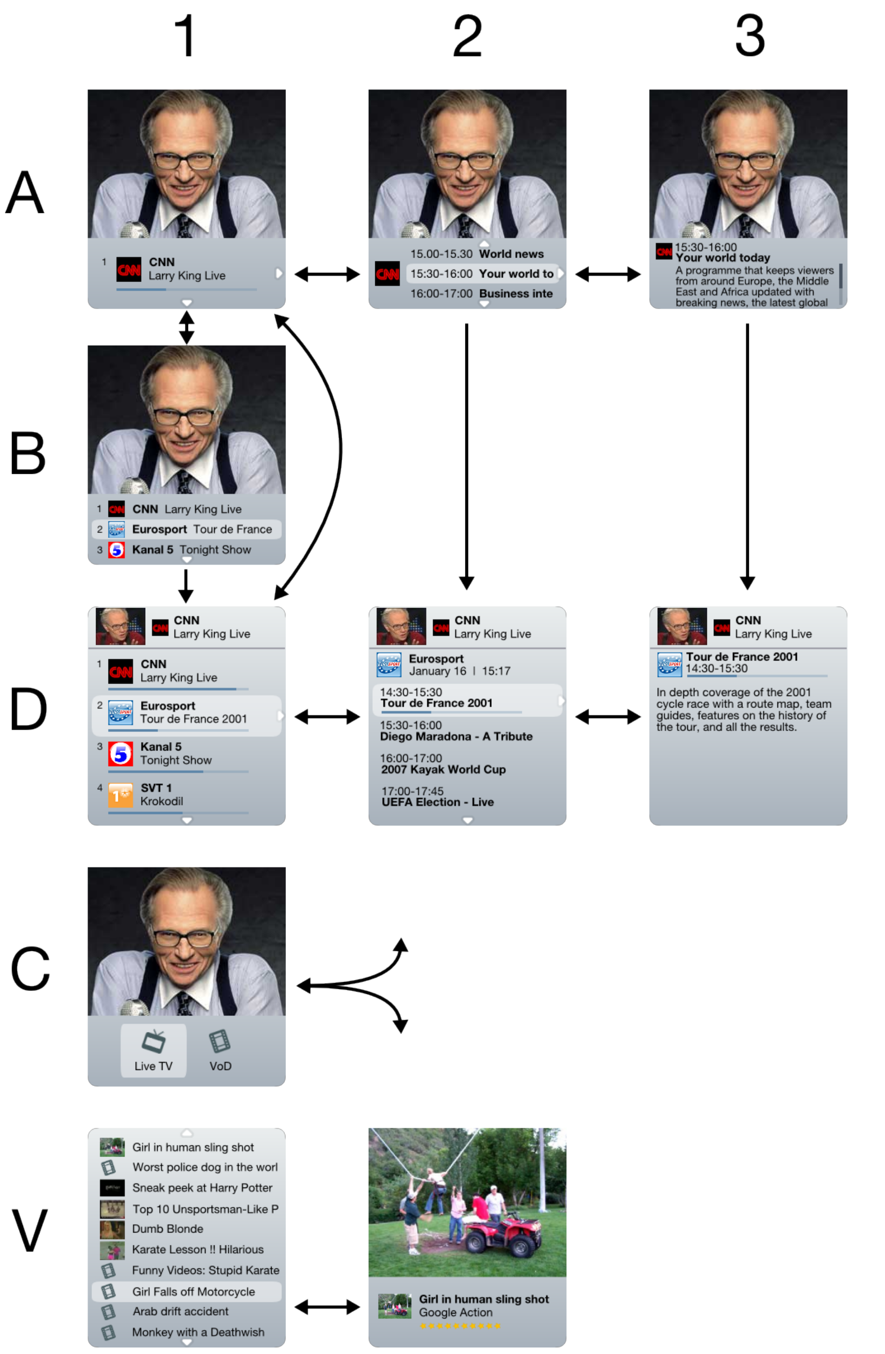


Figure 4.11. View V2. This is the video on demand equivalent to the A1 view. It shows some information about the clip being watched, for example the clip title and the rating in number of stars. Navigate left to go back to the V1 view (without stopping the clip). When the clip has finished playing, you will be automatically transfered back to the V1 view.


1
2
3
A
B
D
C
V
1 CNN Larry King Live
15.00-15.30 World news
15:30-16:00 Your world to
16:00-17:00 Business inte
15:30-16:00 Your world today
1 CNN Larry King Live
2 Eurosport Tour de France
3 Kanal 5 Tonight Show
CNN Larry King Live
1 CNN Larry King Live
2 Eurosport Tour de France 2001
3 Kanal 5 Tonight Show
4 SVT 1 Krokodil
Eurosport January 16 | 15:17
14:30-15:30 Tour de France 2001
15:30-16:00 Diego Maradona - A Tribute
16:00-17:00 2007 Kayak World Cup
17:00-17:45 UEFA Election - Live
Tour de France 2001 14:30-15:30
Live TV
VoD
Girl in human sling shot
Worst police dog in the worl
Sneak peek at Harry Potter
Top 10 Unsportsman-Like P
Dumb Blonde
Karate Lesson !! Hilarious
Funny Videos: Stupid Karate
Girl Falls off Motorcycle
Arab drift accident
Monkey with a Deathwish
Girl in human sling shot
Google Action

A programme that keeps viewers from around Europe, the Middle East and Africa updated with breaking news, the latest global

In depth coverage of the 2001 cycle race with a route map, team guides, features on the history of the tour, and all the results.

Figure 4.12. Navigating between the views in the graphical user interface. The position in the matrix corresponds to the respective view name.

4.3 User input control

How well the user will be able to interact with the application and thus how well the application is perceived, is directly related to how well we can handle user input. For example, if the user expects to use the joystick in order to navigate, but the application is unable to handle this, the result will be a bad user experience. A keypad on a mobile phone usually has twelve keys, digits 0-9 plus the star and the hash key. In addition, there are typically a number of keys that are referred to as the soft keys. The soft keys are used to navigate and interact within the user interface of the phone and often include a joystick or a set of directional keys. As we have shown before, what we typically have control over via key events in JavaScript are the keypad keys, but not the soft keys.

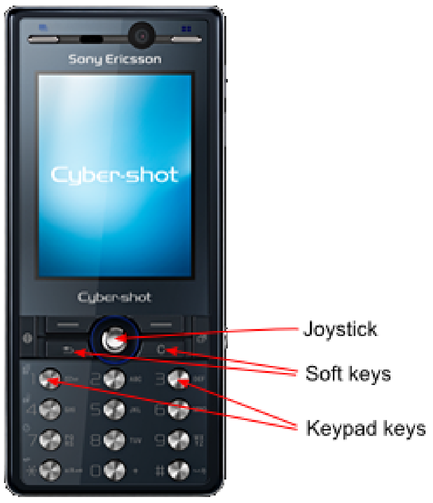
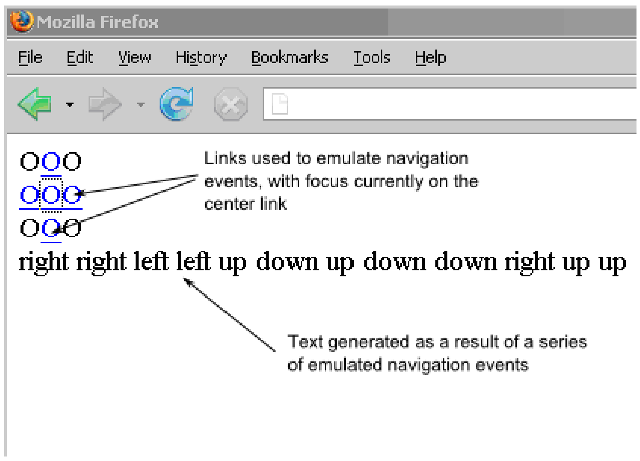


Figure 4.13. The keys on the Sony Ericsson K810i.

To provide the best user experience it is essential for the user to navigate within a web application using the phone's navigator, for example the joystick. We here present a solution based upon emulating key events from the navigator; in order to use it as though it had built-in support for triggering such events.

The navigator is typically designated to navigate within links in a web page. When a link is highlighted as a result of the user moving to that link, an `onfocus` event is triggered. Focus is set by the user navigating to a particular link, but can also be set manually from within the application. By creating a 3x3 matrix of links and setting focus on the link in the center, an `onfocus` event will be triggered by adjacent links when the user navigates in any direction. As soon as any such event is caught, focus is immediately returned to the center link and the application is once again ready to handle user navigation. Note that the matrix of links will probably be invisible to the user.

Our navigation handler is preferably placed within a wrapper object so that we


Mozilla Firefox
File Edit View History Bookmarks Tools Help
OOO
OOO
OOO
Links used to emulate navigation events, with focus currently on the center link
Text generated as a result of a series of emulated navigation events

right right left left up down up down down right up up

Figure 4.14. The 3x3 matrix used to simulate navigation events. An example page viewed in Firefox.

can register a single key handler that will use key events if available or else fall back on this method of emulating events. An argument to the function registering the key handler will be a callback function to be called whenever an event is triggered, just as when event listeners are normally registered in JavaScript. When a key event is triggered, whether real or simulated, the registered callback function is called with a key code as its argument. From within the callback there is no way of determining if the event originated from a real or simulated key event. The way the key handler object handles key events is transparent, not only to the user, but to the rest of the application as well.

The Mobile TV client relies on the described technique for almost all navigation. However, the keypad digits also provide short-hand keys for switching to channels 1 to 9, while as described in section 4.1, the star and the hash key have been assigned to special functions also essential to navigation.

When previously discussing the graphical user interface it was noted that the concept of *views* plays an important role in relation to user input control. The callback from the key handler consists of a single switch clause used to map keycodes onto methods within the current view. For example, if a keycode is received as a result of the user pressing the navigator up, the method `View.current.move_up()` is invoked, thus passing the action to the current view for further processing. Thereby, user input control will be handled by each view individually whenever a view is current and thus visible on the screen.

4.4 Client-server communication

When the web page constituting the whole client is requested as a result of the user navigating to that page, the client's logic and all the graphical components (images) are downloaded as well. After this, all subsequent requests are made from within the client in order to send commands and fetch content from the server. This subsequent communication we denote client-server communication.

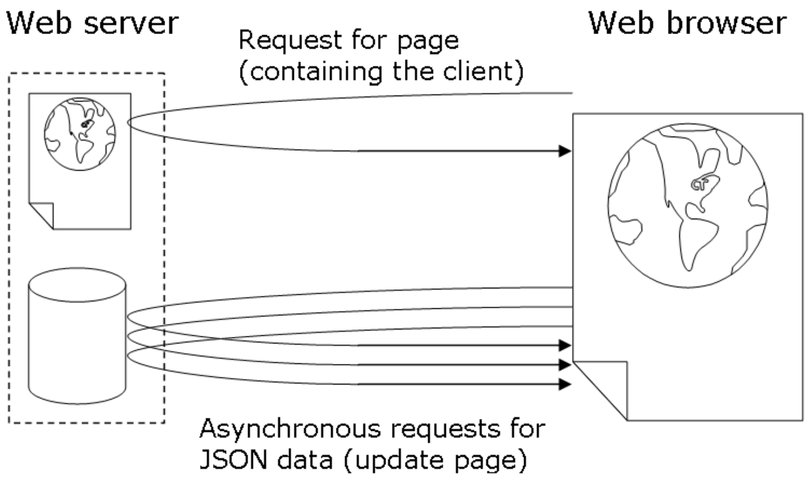


Figure 4.15. The web application is loaded in the browser as a normal web page. All subsequent requests are made by the web application itself.

The first request made by the client is for the server to allocate resources for a new user. If successful, the server will then reply with a full channel list, an RTSP link, and other useful data such as the server time and which channel is currently being shown. Further requests could be, for example, to change channel or poll for interactivity events. The client will also perform a periodic update every other minute to indicate to the server that it is still active.

The server has been extended to be able to reply with JSON-encoded messages instead of with XML as it was originally designed. The reason for using JSON is not only because of the lack of XML support in the Netfront 3.3 browser, but also because of the ease of turning JSON-encoded data into JavaScript objects.

To simplify the calls to the server from within the client, a utility function called `HTTP.getJSON()` has been created. This function takes two arguments, the URL to be retrieved and a callback function. The callback is the function to be called with the loaded data once it has been received and turned into an object as a result of decoding from JSON. Decoding is done using the built-in `eval()` function as we assume that we have secure communication with the server and that this server only provides "safe" data. Thus we save the overhead of using a separate JSON decoder. `HTTP.getJSON()` will use `XMLHttpRequest` if the browser supports it and reads the `responseText` property to get the loaded JSON data. Otherwise, it falls back on the `iframe` technique where a URL is loaded into an invisible `iframe` by setting the `location` property and the data is read as the iframe's contents. Multiple simultaneous requests are supported by using as many hidden iframes as needed.

It is important to be aware of how the browser caches content. If we request data using the same URL all the time, we have to make sure that the data is not cached by the browser. Since the browser caches data based on the URL, only the first request would be made to the server. The subsequent requests would simply be supplied by the cache, if care has not been taken in order to disable caching. A naïve approach may be to alter the URL somewhat between requests, for example to add a counter parameter that would be increased for every request. However, using this method, pages would still be cached, filling up the browser cache and causing items in the cache to be evicted. Since we in principle have no control over how items in the cache are evicted, we may loose items that we would like to have in the cache, for example images or other elements comprising the GUI. Instead, caching is disabled by setting the HTTP headers `Cache-control` and `Expires` appropriately [29].

4.5 Interactivity

One of the great advantages when doing web development is the browsers' ability to render generic content. This has been especially useful when implementing interactivity events in the web-based Mobile TV client. Interactivity events could in principle be everything from a simple textual advertisement banner to a social networking site with team watching and channel chatrooms. By using a visible iframe of varying size, placed on top of the graphical user interface, we can load interactivity events as separate web pages inside the iframe. Thus, anything that can be rendered by the browser can be an interactivity event. We can even load web pages with scripts and have another web application inside the web application, although deeply nesting of iframes may lead to unexpected behavior.

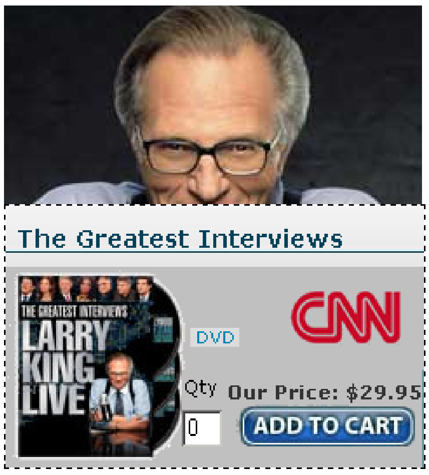


Figure 4.16. An example interactivity event on top of the graphical user interface.

The events are specified on the server with a set of parameters. A simple web page has been created that is used to define a new event and upload its parameters to the server. In fact, it is even possible to enter the contents of the event as HTML code and have it uploaded to the server as well. However, the most common way to specify an interactivity event is to enter a URL from where its markup (and possibly JavaScript code) should be loaded. Other important parameters are: the time and duration when it should be shown on the client; the width, height, and position; and for which channels it is valid.

The client polls the server every minute for the list of interactivity events. Every event has a sequence number and the client always calls the server with the last number seen to prevent the same event from being sent multiple times. The list is actually an array encoded as JSON. Each element in the array is in turn an associative array containing the parameters defining each event. The client has been constructed so that for each event an instance of an interactivity handler object is created with the parameter object passed as argument. The handler objects are independent from the rest of the client and do their own scheduling of events, although the handler reads a global variable to check which channel the user is watching and interfaces with the video object to see if the video has to be repositioned in order to fit the interactivity event. An interactivity handler checks the channel twice, just before an event is about to be loaded and again just before it is shown. This is done to determine if an event is to be displayed on the current channel; the reason it is done twice is because the user could possibly change channels after the event has started loading, but before it has been displayed.

A time parameter `start` denotes the time (on the server) when an event should start loading. Loading of an event is typically done in at most a couple of seconds. If the timing needs to be more exact, one could easily modify the interactivity handler to prefetch the event a short period before it is scheduled, thereby using the `start` parameter to indicate the time when an interactivity event is to be shown. The parameter `stop` could be given either as absolute time when the event is removed or as a duration in seconds from the `start` time using the prefix "+". If `stop` is left out or set to 0 or +0, the interactivity event will not time out, then the event will have to handle its own removal.

Two parameters have been defined to be passed with URLs inside an interactivity event, but are used on the client side only. When a URL containing `STOP-TIMEOUT` is loaded inside the iframe, a timeout set from the `stop` parameter is removed. This is useful for events that we would like to disappear after a while if the user has not interacted with the user interface, but we want to make sure that it is not timing out once the user has started interacting. If the interactivity handler sees the `STOP-INTERACT` parameter inside a URL, it immediately makes the event iframe invisible. The request for that URL will still be made by the browser, making it possible to send data to the server along with it. This could for example be the user's vote in a poll.

Chapter 5

Evaluation

When the concept of a web-based Mobile TV client is presented to someone who has an idea of the technologies involved, there are typically two concerns raised about the feasibility of such implementation. First of all, how good will it be feature-wise, that is, can the client be implemented so that all of the requested functionality is present. Secondly, what will the performance be like, mostly with regards to the end-user experience. It is inevitable that a new client implementation is compared to the already existing ones. In this case, the comparisons have mainly been with the Ericsson Research prototype J2ME client and the commercial Ericsson J2ME client. This chapter evaluates the implemented Mobile TV client based on web technologies, as described in the previous chapter.

5.1 Functionality

What will ultimately limit the functionality of a web application is the fact that there is a web browser in between the application and the user. The browser has its own set of functions that should be available, for example, and as previously discussed, it may have some shortcut keys, which invoke browser functions mapped onto the keypad keys. Luckily, these shortcut keys can in most cases be turned off by the user. However, there are other keys controlling the browser that are hard or impossible to override.

A typical example of this is the back button, used to navigate backwards to previously visited pages. The motive behind disallowing control over the back button from within a web application is that the back button is supposed to always work as the user expects it to work. The only problem is that these expectations vary depending on the context and also vary from user to user. In some cases when a user is navigating through the various views, it would be intuitive to use the back button to "go back". Especially since it works that way in existing Mobile TV clients. For the web client this could have the effect of navigating away from the current page, thus leaving the application. A non-standard compromise between disabling the back button and having the user unintentionaly leave a page was introduced in Internet Explorer 4.0 for Windows and later adapted by other by other browsers. The `beforeunload` event, when registered, triggers immediately after the back button is pressed, presenting the user with a dialog, where he has to confirm leaving the current page. If the `beforeunload` event were implemented in mobile browsers, it could used to lessen the effect of unexpected behavior.

Sometimes the back button will make no sense, for example when we are using the hidden iframe technique to load data for the application, those requests will be added to the page history and when the user presses the back button, these requests will be made again. If such a request was for a channel change, the resulting behavior is probably not what the user expected, as suddenly the channel is changed, but the graphical user interface is not updated accordingly. This particular problem can be prevented by making requests be valid only once, using a sequence number.

There are typically other keys that are assigned to function menus in the browser that would be nice to be able to use in a web application, although they are of less importance. The same keys are used in the other Mobile TV clients based on J2ME, where a developer has complete control over all keys and can assign any functionality to them. The limited control over keys in a mobile web application is usually one of the first drawbacks mentioned when compared to a standard application. However, as our prototype shows, it is possible to overcome most of these limitations so that the web application can co-exist with the web browser.

A true advantage of the web-based Mobile TV client is, as mentioned before, that the application is loaded into the browser as a web page without the user having to install an application. With this thin-client type of functionality it is possible to have themes or skins that are updated weekly or even daily. Different skins or themes can have their own URLs for their own look and feel, but rely on the same JavaScript code for the logic. Updates are hassle free because we know for sure that everyone is always running the latest version (or at least the version that was most recent when they started their session).

The way interactivity events can be handled in the browser environment is really excellent. By relying on the browser's rendering capabilities, interactivity events are defined as independent web pages. This should be compared with the other clients, which use proprietary mark-up based on XML to describe interactivity events and these have to be handled by the application directly. The commercial J2ME client uses an SVG-like format to describe graphics and texts for its interactivity.

5.2 Performance

Because JavaScript is interpreted by the browser directly from the source code, performance could become an issue for a mobile web application. A mobile phone typically has limited computational power, and in this case we also have a streaming player running alongside the client logic. This will of course affect performance even further.

It is necessary to write code that works within the constrains of its operation, but delivers the performance that is expected. This often means that you have to write efficient code. Different design choices may have a great impact on performance.

The Mobile TV web application has an acceptable performance when navigating in the user interface and switching between views, but not be compared to a J2ME client, which is superior in this area. The prototype J2ME client has animated menus (when the video is not playing), something that is currently not possible in the web client. However, improved SVG support in the web browser may enable the use of animations, since SVG animations perform much better than animations using JavaScript and CSS. (Animations using JavaScript and CSS is common on the desktop, but they require too much performance for the mobile phones currently available.)

It is difficult to do good quantitative time measurements of a web application running in the browser of a mobile phone, since there is no interface available for doing this kind of timings. Any attempt to add JavaScript code for doing time measurements would add overhead to the application, thus invalidating any measured times. An alternative would be to film the mobile phone screen using a high-speed camera with a constant frame rate and then counting the number of frames between two occurrences that one wish to measure. However, it has not been prioritized in this project.

There are some areas in which the web application performs better than the J2ME clients. For example, the web client is significantly faster when switching channels. In the J2ME prototype client, switching between two channels is typically about 4 seconds compared to the web client where the same operation takes on average slightly less than 3 seconds (and sometimes as little as 2 seconds). This is due to the browser being more efficient in setting up HTTP connections than the code in the Java environment. Most of the time being spent switching channels is for the new content to propagate.

Additionally, when playing video-on-demand clips by changing the RTSP source, the web client has the best performance. While watching a clip in the web client, starting a new clip by having the player initiate a new RTSP session takes about 7 seconds until the new clip is playing (of which 4 seconds are for buffering). Doing the same thing in the J2ME prototype client takes between 10 and 11 seconds.

There is one area in which the web client vastly outperforms the J2ME prototype client. In the web client, the user can instantly (in less than a second) switch between full screen video and full width video, instead of having to wait approximately 7 seconds as in the J2ME prototype client.

Chapter 6

Conclusions and future work

The foremost conclusion to be drawn from this thesis project is that the web-based client implementation came out very well, better than most people expected. The greatest cause of excitement seems to be how well one can mimic the look and feel of, for example, a J2ME application. A contributing factor to this has probably been that the web application was based on an existing user interface design. It is possible that the user interface would be designed differently if targeted specifically for a web application that would run inside a browser. Because the GUI is nearly identical to that of the comerical J2ME client, the problems related to the back button are made worse because of users being used to navigating with that button. A future enhancement is to make requests for channel change valid only once, by maintaining state on the server. This would at least prevent an obviously incorrect behavior of the client, if a user happens to press the back button.

There are some other issues that need to be resolved before we could recommend deployment of a commercial Mobile TV web application. The most problematic aspect is probably that Smart-Fit on Netfront 3.3 is unconditionally applied when it is enabled, considering that the default setting is On. The default setting for browser shortcuts is also On in that browser, which means that there are two default settings that the user has to change before the application will work correctly. The fact that there are few terminals yet supporting a streaming video plugin has to be considered as well.

Another major issue with the Sony Ericsson phones running Netfront concerns the video plugin. The screen is abruptly turned off when the user is "idle", even though the video is playing. When the screen is turned off, all execution of JavaScript code is halted, effectively disabling interactivity events and maintenance routines running in the background. JavaScript is also disabled when the video plugin is in full screen mode. The limited JavaScript API towards the video plugin, with `playSource()` as the only callable method, should be extended to allow for more control. For example, there is no way for a web application to know if the video has stopped playing (besides asking the server, of course).

Despite the minor issues, this project has really shown that a web application can replace a local application in the case of Mobile TV. The web application can support the required functionality and the performance is comparable and sometimes even better than a J2ME application.

Possible future work includes extending the video-on-demand functionality with more client features, but perhaps more interesting, build a server-side solution where RTSP links with associated meta data are harvested from the Internet and put into a database, later to be used by the client. This could include having an RTSP proxy with live transcoding and caching of video content.

The web-based Mobile TV client should be extended to support more terminals. The client should be tested on the phones that we know have a streaming plugin (i.e. recent Samsung phones with Netfront). Future work could examine phones with an advanced web browser (such as Opera), without a streaming video plugin, but with a streaming video player, to see if they could still make use of the client. In this case video content would have to be opened externally by the video player by passing it RTSP links from the web browser.

Bibliography

[1] Christoffer Andersson, Daniel Freeman et al., Mobile Media and Applications - From Concept to Cash, John Wiley & Sons, January 2006, pp 166-167

[2] C. Carlsson, P. Walden, Mobile TV - To Live or Die by Content, HICSS 2007, January 2007, p. 51

[3] C. Forrester, Standards war looms [Mobile TV], IEE Communications Enigneer, Volume 3 Issue 5, Oct./Nov. 2005

[4] D. Sandham, What next for Mobile TV?, IEE Review Volume 52 Issue 3, March 2006

[5] V. Ollikainen, C. Peng, Integration of DVB-T into DVB-H transmission systems, VTT Technical Research Centre of Finland, http://users.tkk.fi/ pcy/IBC_5289_final.pdf, Last modified: 2006-11-01

[6] Ian S. Graham, The HTML Sourcebook, John Wiley & Sons, March 1995

[7] Eric A. Meyer, CSS: The Definitive Guide Third Edition, O'Reilly Media, November 2006

[8] David Flanagan, JavaScript: The Definitive Guide Fifth Edition, O'Reilly Media, August 2006

[9] M. Mahemoff, Ajax Design Patterns, O'Reilly Media, June 2006

[10] David Crockford, JSON: The Fat Free Alternative to XML, http://www.json.org/fatfree.html, Last modified: 2006-12-18

[11] David Crockford, RFC 4627: The application/json Media Type for JavaScript Object Notation (JSON), IETF Network Working Group, July 2006

[12] David Eisenberg, SVG Essentials, O'Reilly Media, February 2002

[13] Kurt Cagle, SVG Programming: The Graphical Web, Apress, 2002

[14] Richard Leggett, et al., Flash Applications for Mobile Devices, Apress, December 2006

[15] Ian Hickson (Editor), Web Applications 1.0 Working Draft, WHATWG, http://www.whatwg.org/specs/web-apps/current-work/, Last accessed: 2007-03-01

[16] D. Kristol and L. Montulli, RFC 2109: HTTP State Management Mechanism, IETF Network Working Group, February 1997

[17] ACCESS Co., Ltd., NetFront Browser for Mobile, http://www.access-company.com/products/netfrontmobile/index.html, Last accessed: 2007-07-03

[18] Tolga Capin (Editor), Mobile SVG Profiles: SVG Tiny and SVG Basic, W3C Recommendation, 14 January 2003, http://www.w3.org/TR/SVGMobile/

[19] Opera Software, Products featuring the Opera Mobile Browser, http://www.opera.com/products/mobile/products/, Last accessed: 2007-02-19

[20] The WebKit Open Source Project, http://webkit.org/, Last accessed: 2007-07-03

[21] Apple Inc., Safari, http://developer.apple.com/internet/safari/, Last accessed: 2007-07-03

[22] KDE e.V., Konqueror - Web Browser, File Manager and more!, http://www.konqueror.org/, Last accessed: 2007-07-03

[23] The Minimo Project, Minimo - A small, simple, powerful, innovative, web browser for mobile devices, http://www.mozilla.org/projects/minimo/, Last accessed: 2007-07-03

[24] Mozilla Foundation, Mozilla Layout Engine, http://www.mozilla.org/newlayout/, Last accessed: 2007-07-03

[25] Abe Fettig, Twisted Network Programming Essentials, O'Reilly Media, Inc., October 20, 2005

[26] Trygve Reenskaug, Models - Views - Controllers, Xerox PARC, 10 December 1979

[27] The Apache Software Foundation, Batik SVG Toolkit, http://xmlgraphics.apache.org/batik/, Last published: 2007-03-30

[28] Wireless Application Protocol Forum, Ltd., WAP WML Specification, 19 February 2000, http://www.openmobilealliance.org/release_program/docs/Browsing/V2_1-20050614-C/WAP-191-WML-20000219-a.pdf

[29] R. Fielding, et al., RFC 2616 section 13: Caching in HTTP, The Internet Society, June 1999, http://www.w3.org/Protocols/rfc2616/rfc2616.html, Last accessed: 2007-07-03

List of Abbreviations

3GPP	3rd Generation Partnership Project
Ajax	Asynchronous JavaScript and XML
API	Application Programming Interface
CSS	Cascading Style Sheets
DAB	Digital Audio Broadcasting
DOM	Document Object Model
DVB-H	Digital Video Broadcasting - Handheld
DVB-T	Digital Video Broadcasting - Terrestrial
ECMA	European Computer Manufacturers Association
EPG	Electronic Program Guide
ETSI	European Telecommunications Standards Institute
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	HyperText Transport Protocol
J2ME	Java 2 Platform, Micro Edition
JSON	JavaScript Object Notation
MBMS	Multimedia Broadcast Multicast Service
MVC	Model-View-Controller
OSI	Open Source Initiative
RTP	Real-time Transport Protocol
RTSP	Real Time Streaming Protocol
SGML	Standardized Generalized Markup Language
SVG	Scalable Vector Graphics
T-DMB	Terrestrial Digital Multimedia Broadcasting
uDOM	Micro Document Object Model
URL	Uniform Resource Locator
VoD	Video on Demand
W3C	World Wide Web Consortium
WAP	Wireless Application Protocol
WHATWG	Web Hypertext Application Technology Working Group
WML	Wireless Markup Language

List of Abbreviations

XHTML	eXtensible HyperText Markup Language
XML	eXtensible Markup Language

COS/CCS 2007-21