HELSINKI UNIVERSITY OF TECHNOLOGY
Faculty of Electronics, Communications and Automation
Degree Programme of Communications Engineering

# The Web as a Runtime in Mobile Context

Master's Thesis

Anssi Kostiainen

Department of Media Technology
Espoo 2008

| HELSINKI UNIVERSITY OF TECHNOLOGY | ABSTRACT OF MASTER'S THESIS |
|---|---|

Faculty of Electronics, Communications and Automation
Degree Programme of Communications Engineering

**Author:** Anssi Kostiainen

**Title of thesis:**
The Web as a Runtime in Mobile Context

| | | | |
|---|---|---|---|
| **Date:** | April 15, 2008 | **Pages:** | xiv + 143 |
| **Professorship:** | Interactive Digital Media | **Code:** | T-111 |
| **Supervisor:** | Professor Petri Vuorimaa | | |
| **Instructor:** | Ph.D. Kimmo Raatikainen | | |

Web technologies were initially designed to facilitate the creation of static web pages. However, the ubiquity of the web browser has motivated the use of the same technologies as a basis for desktop-style applications which are executed within the web browser and have their characteristics such as high interactivity.

Despite the popularity of web applications, there exists various problems due to the fact that established web technologies were not specified with applications in mind. In addition, the constraints introduced by mobile devices challenge the ubiquity of such applications. On this account, new platforms have emerged extending the capabilities of web browsers.

In this thesis, the requirements for client-side web applications in mobile context are synthesized. Moreover, a taxonomy for web applications is drawn and client-side web technologies and major software platforms relevant to the client-side web applications are discussed. Furthermore, an application concept implementation developed using web technologies leveraging the capabilities of the major mobile platform is presented and evaluated against the defined requirements.

The evaluation revealed various problems related to limited expressiveness of web technologies, interoperability and platform functionality. Regardless, the implementation provided a level of functionality comparable to that of native applications.

**Keywords:** Web, web browser, platform, runtime environment, mobile devices, S60, W3C, XML, JavaScript, Ajax
**Language:** English

TEKNILLINEN KORKEAKOULU DIPLOMITYÖN TIIVISTELMÄ
Elektroniikan, tietoliikenteen ja automaation tiedekunta
Tietoliikennetekniikan koulutusohjelma

**Tekijä:** Anssi Kostiainen

**Työn nimi:**
Web ohjelmistoalustana mobiililaitteissa

| | | | |
|---|---|---|---|
| **Päiväys:** | 15. huhtikuuta 2008 | **Sivumäärä:** | xiv + 143 |
| **Professuuri:** | Vuorovaikutteinen digitaalinen media | **Koodi:** | T-111 |
| **Työn valvoja:** | Professori Petri Vuorimaa | | |
| **Työn ohjaaja:** | FT Kimmo Raatikainen | | |

Web-teknologiat kehitettiin alun perin kuvaamaan staattisten web-sivujen sisältöä. Web-selainten suosion vuoksi samoja teknologioita hyödynnetään nykyisin myös sovellusten toteuttamiseen käyttäen web-selainta niiden suorittamiseen vuorovaikutteisesti.

Web-teknologioiden suosiosta huolimatta ne sisältävät useita ongelmia ohjelmistojen toteuttamisen näkökulmasta. Lisäksi mobiililaitteiden rajoitukset tekevät kaikilla laitteilla toimivien sovellusten toteuttamisesta haasteellista. Tämän vuoksi uusia ohjelmistoalustoja on kehitetty ratkaisemaan web-selainten asettamia rajoituksia.

Tässä diplomityössä koostetaan vaatimukset mobiililaitteissa toimiville web-sovelluksille. Tämän lisäksi web-sovelluksille luodaan luokitusjärjestelmä ja tärkeimmät web-sovelluskehitykseen liittyvät web-teknologiat ja ohjelmistoalustat esitellään. Esimerkkisovellus toteutetaan käyttäen web-teknologioita ja hyödyntäen ohjelmistoalustan ominaisuuksia. Lisäksi esimerkkisovellus arvioidaan määritettyä vaatimusmäärittelyä vastaan.

Arviointi paljasti lukuisia haasteita, jotka liittyivät web-teknologioiden puutteelliseen ilmaisuvoimaan, yhteentoimivuuteen ja ohjelmistoalustan toiminnallisuuteen. Tästä huolimatta esimerkkisovellus toteutti sille asetetut vaatimukset työpöytäohjelmistotasoisesta toiminnallisuudesta.

**Avainsanat:** Web, web-selain, ohjelmistoalusta, virtuaalikone, mobiililaitteet, S60, W3C, XML, JavaScript, Ajax
**Kieli:** Englanti

# Acknowledgements

Thinking of writing a thesis and actually starting to write one – let alone getting one done – are obviously different things. I would like to thank Timo Ali-Vehmas for kick-starting this project and giving me an opportunity to write my thesis during my work at Nokia.

I want to thank Art Barstow for helping me with the W3C topics. I would also like to thank Guido Grassel and his team for their inspiring research. Special thanks go to Mikko Honkala for finding time for discussions and constructive comments.

My gratitude goes to Mikko Pohja for instructing the last mile of the thesis by finding time to comment my work with such a short notice and for ideas which gave the last finishing touches. I would also like to thank my supervisor Petri Vuorimaa for pointing me to the relevant research early on.

My former instructor and colleague Kimmo Raatikainen helped me formulate the foundations and the initial structure of the thesis. Kimmo gave me advice on scientific writing and kept me on the topic. I am deeply grateful Kimmo chose to share some of his limited time with me during the last year. The memory of Kimmo's endless energy motivated me to finish this thesis. I sincerely wish Kimmo had had the opportunity to see this thesis to materialize.

Finally, I would like to thank my beloved Elisa, who stood by me even if the writing process seemed like an endless journey.

Helsinki, April 15, 2008

Anssi Kostiainen

# Glossary

| | |
|---|---|
| AIR | Adobe Integrated Runtime, a runtime environment for RIAs developed by Adobe. |
| Ajax | A programming technique used in web application development. |
| Android | A mobile phone platform based on Linux kernel developed by Google. |
| API | Application Programming Interface, a convention for accessing functionality of a computer program. |
| Atom | A pair of related standards for feed syndication and publishing. |
| ARIA | Accessible Rich Internet Applications Suite, a set of specifications defining how to make advanced features of dynamic web content accessible. |
| CSS | Cascading Style Sheets, the language used to describe the presentation of structured documents in the Web. |
| Declarative language | A programming language, which describes what some functionality is like, rather than defining how to implement it. |
| DOM | Document Object Model, an API and a data model for representing (X)HTML and XML documents. |
| ECMAScript | A scripting programming language, standardized by Ecma International in the ECMA-262 specification. |
| GWT | Google Web Toolkit, an open source Java framework that allows creating Ajax applications in Java. |
| Handheld device | See Mobile device. |
| HTML | Hypertext Markup Language, the predominant markup language for the creation of web pages. |
| HTTP | Hypertext Transfer Protocol, the main communications protocol of the Web. |
| JavaFX | A Suite of technologies and a runtime environment for web application development by Sun Microsystems. |
| Java ME | A subset of the Java platform for constrained devices. |
| JavaScript | A scripting language often used for client-side web development. A dialect of the ECMAScript. |
| JSON | JavaScript Object Notation, a lightweight format for interchanging data. |

| | |
|---|---|
| Layout engine | Software that takes web content (such as HTML, XML or image files) and formatting information (such as CSS) and displays the formatted content on the screen. |
| Markup language | A language containing annotations which are machine-readable and typically embody human-readable text. |
| Middleware | Software between the application software and the operating system. |
| Mobile application | A generic term for a software system operating on a mobile device. |
| Mobile device | A pocket-sized computing device, typically utilizing a small visual display screen for user output and a miniaturized keyboard for user input. Typical mobile devices are smartphones, mobile phones and PDAs. |
| MVC | Model-View-Controller, a commonly used architecture in GUI applications. |
| .NET Compact Framework | A subset of .NET Framework for mobile and embedded devices. |
| .NET Framework | A development platform for Windows operating system. |
| OPML | Outline Processor Markup Language, an XML-based aggregate format. |
| PDA | Personal digital assistant, an electronic device which can include some of the functionality of a computer. |
| Platform | A set of services, technologies and APIs on which additional software and applications are constructed. |
| Procedural language | Contrary to declarative language, specifies the steps the program has to take to reach the desired state. |
| REST | Representational State Transfer, an architectural style for distributed systems embodied in the design of HTTP. |
| Rendering engine | See Layout engine. |
| RSS | A family of feed formats used to publish frequently updated content in the Web. |
| RIA | Rich Internet Application, a web application that has the features and functionality of a traditional desktop application. |
| Runtime | See runtime environment. |
| Runtime environment | Software which manages the execution of application code and provides various services to the application. |
| S60 | A software platform for mobile phones that uses Symbian OS. |
| S60 Web Runtime | A runtime environment part of the S60 for executing applications built on core web technologies. |
| S60WebKit | A port of the WebKit to the S60. |
| Silverlight | A proprietary runtime environment and a browser plugin for running RIAs developed by Microsoft. |
| SMIL | Synchronized Multimedia Integration Language, an XML-based language for creating interactive multimedia presentations. |

| | |
|---|---|
| SVG | Scalable Vector Graphics, an XML-based markup language for defining vector graphics. |
| Symbian OS | A proprietary operating system designed for mobile devices. |
| UI | User interface, provides input and output facilities to the user of a computer program. |
| URI | Uniform Resource Identifier, a compact character string identifying or naming a resource. |
| User agent | A program that accesses web resources for any purpose. |
| Virtual machine | A software implementation of a computer that executes programs. |
| W3C | World Wide Web Consortium, the main international standards organization for the World Wide Web. |
| Web application | An application that is built on web technologies, typically accessed via the Web over a network such as the Internet using a web browser. |
| Web Browser for S60 | A web browser for the S60 based on WebKit developed by Nokia. |
| WebKit | An open source application framework and a rendering engine. |
| WHATWG | Web Hypertext Application Technology Working Group, a group of browser vendors and other interested parties working on the next generation of HTML. |
| WICD | A Web Integration Compound Document, a specification which ties the core presentational web technologies together. |
| Widget | A class of client-side web applications. |
| WRT | S60 Web Runtime, a runtime platform for S60 used to execute applications built on standard web technologies. |
| XAML | Extensible Application Markup Language, a proprietary XML-based markup language for defining UI elements developed by Microsoft. |
| XBL | XML Binding Language, a declarative language for binding XML documents to each other. |
| XForms | A specification defining the next generation of XHTML forms. |
| XHTML | Extensible Markup Language, HTML reformulated as an XML language. |
| (X)HTML | Un umbrella term referring to both HTML and XHTML. |
| XML | Extensible Markup Language, a general-purpose specification for creating custom markup languages. |
| XUL | XML User Interface Language, an XML markup language developed by the Mozilla for defining application layouts. |
| XULRunner | A runtime environment for XUL applications developed by Mozilla. |

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

The World Wide Web – the Web in short – is a system built on top of the Internet which has been a phenomenal success since the late nineties. In a short period of time, the Web has developed from its initial form used at CERN[1] [9] for managing information using simplistic web pages into a platform of interactive services. For example, today the Web is used for online shopping, games, multimedia applications and maps among others. Most importantly, the technical evolution of the Web has happened inconspicuously of a regular web user thanks to the extensible architecture of the Web. On the other hand, the way people access information today has undergone a revolution – URLs are printed on everything, *google* has become a verb and the web browser has morphed into a window encompassing information earlier scattered around newspapers, dictionaries and bulletin boards.

While the Web has been evolving, mobile devices have developed from simple devices used for voice communication only into pocket-sized computers which contain enough processing performance and capacity, Internet connectivity and a modern operating system to be first-class citizens of the Web. At present there are over 3 billion mobile devices – more than web users altogether – and within 2-3 years the amount is expected to increase by at least a billion [178]. At the same time the use of the Web on mobile devices is increasing rapidly. For example in Japan, there exists almost equal amount of people using the Web on a mobile device and on a desktop in 2007 [37].

The most distinctive characteristics of web applications compared to traditional applications are that they are built on device-independent web technologies and are accessed via the Web over a network such as the Internet in an on-demand fashion which requires no installation of software as the web

[1]The European Organization for Nuclear Research

browser acts as an ubiquitous client. Traditionally, the browser has taken the role of a thin client in web applications and has been only used for interpreting the user interface whereas the program logic has resided on the web server. However, recent requirements for web applications, especially higher interactivity, have shifted the interest towards client-side and the web browser as an application runtime environment.

Client-side web technologies are by far the most known of all languages used for developing applications [38, 84]. Today, they are in a major role in delivering rich user experience of today's web applications and services to the users. The ubiquity of web technologies has enabled web applications to establish a firm foothold on areas previously dominated by traditional application environments such as Windows [132]. This implies that the associated development, deployment and delivery models are versatile and appropriate for most use cases. Despite the success, the area of web applications has not been adequately served by the web technologies originally designed for thin client approach. As a consequence, alternative platforms extending the web browser have emerged and new ones are being developed. The approaches of these platforms vary from device dependent to theoretically device independent paradigms. Advantages of device dependent approaches are in their better performance and deeper integration with the underlying platform which enable e.g. richer graphics capabilities. Regardless, developing device dependent applications incur a significant cost as the same applications do not typically run across multiple platforms.

Lately, improvements have been suggested to existing web technologies that facilitate authoring of web applications. As the same building blocks of the web applications are also supported in modern mobile devices, it is reasonable to assess the state of web technologies as a platform of web applications for mobile devices and identify the areas which still need further development. If no advances in web technologies are made that facilitate building web applications for mobile devices, it can be expected that disruptive platforms will emerge, hindering the uptake of the Web as a common runtime for applications.

## 1.1 Problem Statement

*The objective of this thesis is to assess the feasibility of client-side web technologies in mobile devices, and especially, their applicability for implementing applications that are today typically developed using device dependent approaches. More specifically, the feasibility of a runtime platform for said* *applications – the S60 Web Runtime [51] – is addressed from the viewpoint of generic software quality, end-user and language requirements by implementing a prototype application and evaluating it against the aforementioned requirements.*

## 1.2 Background

World Wide Web Consortium (W3C) is an international consortium developing standards and guidelines for the Web. Its work is organized into activities, of which the Rich Web Clients Activity is the most relevant within the scope of client-side web technologies and web applications. It aims to improve the client-side experience of web technologies which are used as building blocks for web applications. This includes formats and application programming interfaces (APIs) used for application development. In addition to W3C, other parties have also actively contributed to the development of the web standards which facilitate building more interactive web applications. The Web Hypertext Application Technology Working Group (WHATWG) initially formed by web browser vendors in 2004 has been developing the existing core web standard, the Hypertext Markup Language in a backwards compatible manner to cater for web applications. The specification named HTML 5 [98] was brought to W3C in 2007 and the work is still on-going. New additions in HTML 5 include, for example, much improved graphics capabilities, standard way to embed video and audio elements into web content and store data persistently on the client-side. Related to rich graphics capabilities, Apple has proposed new extensions to the established Cascading Style Sheets specification which is the standard used to describe the presentation of web content. The extensions enable effects familiar from rich graphical toolkits without proprietary extensions commonly used today to be implemented using standard web technologies and bring the presentation capabilities of web applications on par with their native counterparts.

The fundamental constraint of web applications, the lack of offline support, has been also recently addressed. Google has published an extension to the web browser which enables web applications to function offline, which in turn, enables new use cases, such as using web applications instead of traditional applications in places where Internet connectivity is not available. To realize use cases where existing web standards have not provided solutions, many big vendors such as Microsoft and Adobe have released new platforms which promise users more seamless experience compared to web applications implemented in standard web technologies used via standard web browser.

These web runtime environments aim to differentiate by providing better integration with the underlying operating system, richer graphics capabilities and less restricted methods for interacting with the application. However, as a downside these approaches are not as ubiquitous as the web browser and require additional extensions restricted to certain operating systems or devices. The openness of many of these approaches is also disputable.

The above-mentioned developments have been first brought to the desktops, but mobile devices have been following the same path. Today, most sophisticated mobile devices utilize the same components in their web browsers as popular desktop browsers, which should bring them on par with the desktop browsers in standards compliance. Regardless, the existing web standards do not take mobile device specifics into account. This has spawned alternative platforms for mobile devices which leverage web technologies but extend the web browser to provide better integration with the underlying platform for more seamless user experience. An example of such a platform is Nokia's S60 Web Runtime [51] which extends the web browser. Furthermore, new players such as Google and Apple have entered into the mobile business bringing operating systems closely resembling desktops from the technology compliance point of view. This has been a notable difference compared to the traditional approach of the mobile market in which more restricted subsets of technologies have commonly been used.

There is significant growth potential for the Web as an application platform in mobile context. Finnish studies conducted in 2006 concluded that 60% of the traffic in smartphones in Finnish mobile networks was generated by Hypertext Transfer Protocol (HTTP) which is the protocol used to convey information on the Web [121]. A notable observation was that the web browser was the main source for this traffic. In addition, recent improvements in network speeds and browser design together with declining cellular data costs have catalyzed the absolute traffic volumes to three-to-four fold growth from 2005 to 2006 according to the same studies.

Although Finland may not reflect the global state of mobile usage, it arguably is a good indicator of the trend how people may be using their mobile devices within two to three years on a global scale, presupposing capable mobile devices and networks will be more universally available in the coming years. The results from the above-mentioned study imply, that web technologies will play a significant role in mobile context and suggest that there is an untapped growth potential beyond traditional browsing on mobile devices.

## 1.3 Scope of the Thesis

The scope of this thesis is limited to web technologies related to developing client-side web applications. The main focus is on open web standards developed by World Wide Web Consortium (W3C). Proprietary approaches are evaluated in case there exists no open standard fulfilling the key requirements for web applications. The main platforms utilizing web technologies are discussed with a focus on platforms for mobile devices. Based on prior literature, the most suitable platform compliant with the requirements is selected as the basis for concept implementation and its capabilities are evaluated in more detail.

## 1.4 Methodology

The methodological approach of this thesis is characterized by the following more detailed objectives:

1. Synthesize a list of key requirements for client-side web applications consisting of generic quality requirements for a software product, user requirements related to interaction patterns and requirements specific to the interaction and multimedia capabilities of the web technologies and related platforms. The requirements must derive from the existing research and publications related to guidelines, use cases and requirements within the scope of the thesis.

2. Review the current core web technologies, their evolutionary versions and major platforms used to create client-side web applications. In addition, examine the major emerging web technologies that address the key requirements where existing core web technologies do not provide feasible solutions.

3. Realize a concept implementation of a feed reader application which leverages device and platform capabilities and is implemented in core web technologies only.

4. Evaluate the concept implementation against the defined key requirements. Based on the analysis, identify gaps in the existing web technologies and provide recommendations for future developments related to web standards, web browser extensions and web runtime environments.

## 1.5 Structure of the Thesis

The structure of the thesis is the following:

In Chapter 1, the research area is introduced and a problem statement for the thesis is formulated. In addition, related work is discussed and detailed objectives for the research are elaborated.

In Chapter 2, the existing body of knowledge is reviewed related to requirements for web applications in mobile context and the key requirements are synthesized.

In Chapter 3, the web architecture is introduced, core web technologies, their evolutionary versions and emerging web technologies relevant to the scope of the thesis are elaborated.

In Chapter 4, major web application categories are discussed, followed by a review of the major software platforms related to web applications and web technologies within the scope, including operating systems and middleware. Finally, the platforms utilizing web technologies are classified.

In Chapter 5, the concept implementation realized as the experimental part of the thesis is discussed.

In Chapter 6, the concept implementation is validated by evaluating it against the key requirements. In addition, alternative approaches to identified problems proposed by emerging web technologies are discussed where applicable.

In Chapter 7, the results of the thesis are discussed and interpreted in the light of earlier research. In addition, the key findings are summarized.

In Chapter 8, conclusions of the thesis are drawn and suggestions for further research are given.

# Chapter 2

# Requirements for Web Applications

First in this chapter, typical use cases for web applications, especially in mobile context, are presented. Next, common non-functional requirements for such applications are discussed. Furthermore, the existing body of knowledge aimed at web authors is analyzed by reviewing the major guidelines, best practices and interaction design patterns within the scope of the thesis. Finally, a summary of key requirements for web applications built entirely in client-side web technologies is drawn taking into account the needs of the end-users and developers of such applications as well as implications of constraints introduced by mobile devices themselves.

## 2.1 Use Cases for Web Applications

Use cases describe the interaction between the user and the system consisting of a sequence of simple steps which together aim to capture the functional requirements of a system. Use cases are considered an integral part of a successful design process [137]. In the context of this thesis, a web application is defined as an application built on standard web technologies communication over the Internet using standard protocols. Typically, a web browser or similar software available on multiple platforms supporting standard web technologies is used to run such web applications. Use cases have been the starting point also while designing the latest version of core language of the Web, the Hypertext Markup Language (HTML). When the foundations for the latest version of this language were laid, it was stated in [77] that: *"every* *feature that goes into the Web Applications*[1] *specifications must be justified by a practical use case."*

In this section, selected use cases related to web applications published by the W3C are reviewed. Within the scope of W3C, there exists such documents describing use cases and high-level requirements for declarative formats for applications and user interfaces [193], more specific requirements for small client-side Web applications called widgets [26, 25]. In addition, use cases for compound documents provides an extensive list of high-level use cases for rich multimedia content [4, section 1.2].

Furthermore, multimodal interaction, that is applications in which one can interact using multiple modes of interaction, for instance, using speech instead of key presses for input, were described in [28]. Core presentation characteristics use cases cover aspects how to adapt content to a specific presentation capabilities [81]. Use cases for evaluating the potential benefits of an alternative for an Extensible Markup Language (XML) in mobile devices is discussed in [36, section 3.8].

In the context of use cases in [4, section 2.2], web applications running within user agent or within another host application, which have some form of programmatic control on the client are covered. Examples of such use cases are presented below:

**Reservation system** An interactive service allowing users to make reservations by using a graphically-rich user interface. This includes, for example, components such as calendars which aid users in selecting proper dates. In [95] similar use case is extended to cover mobility needs where using a multimodal capable mobile device voice is used as both the input and the output mechanisms to interact with the application.

**Order entry system** An application used on the go to process orders for goods and services. Characteristically such tasks are repetitive and done in a rush which emphasizes the usability and efficiency requirements for such application.

Resident applications are defined as applications partially residing on a device [4, section 2.3]. Examples of such applications include:

**Communication** Tasks such as email, instant messaging and access to device functions to make a call, send a text message or interact with an

[1]The name of the HTML 5 specification at the time.

address book all belong to the communication category. These features are typical of a modern mobile device with network capabilities.

**Information widget** A small application typically part of the device UI in mobile devices which is dynamically updated and displays contextual data related to e.g. user's location and the time of the day. Examples include weather forecasts, special offers and news tickers.

Content authoring, aggregation and deployment use cases include applications for content creation, management and distribution, such as: [4, section 2.3]

**Personal content creation** Refer to user created content that may be shared with others and combined with other content. As an example, user may want to attach a descriptive message to an image he has shot.

**Personal content management** Encompasses activities such as storing and viewing user's personal media such as images, audio, video and text. In addition, such a system should adapt the content for different kind of devices and allow sharing the content with other users.

**Blogs** Web logs or blogs, which are personal diaries published online, are nowadays used for a plethora of communication tasks. They mainly consist of text but may combine other media formats such as images and video as well. The author has an ability to write posts whereas readers can, in addition to passively reading, comment on posts. Typically the user has an ability to subscribe to an aggregate of latest posts, known as web feeds.

**Content aggregation** A class of applications which combine content from various sources together[2]. Typical examples include web portals which provide different delivery channels with varying content. For example, the users of more constrained mobile devices may be offered with smaller images to reduce download times.

The wide spectrum of use cases imply that they set diverse requirements for user interfaces. In the next section, the user interface aspects are discussed, especially focusing on the needs related to mobile devices.

[2]Another term commonly used for such applications is a mashup.

## 2.2 User Interfaces in Computing

The focus of this thesis is on client-side web technologies. These technologies are also used as core building blocks for user interfaces of web applications. In human-computer interaction (HCI), *user interface* (UI) refers to the information (e.g. graphics, text) the program presents to the user as a response to the control sequences (e.g. keystrokes, mouse movements) the user employs to control the program. Today, the most common types of user interfaces are *graphical user interfaces* (GUIs) of desktop operating systems such as Windows and *web-based user interfaces* which are realized within the web browser window.

### 2.2.1 User Interaction Models for Web Applications

The most popular *interaction model* used in traditional desktop application GUIs is WIMP (Windows, Icons, Menus and Pointing). As the name suggests, it revolves around graphical UI elements such as windows, icons and menus which can be interacted with using a pointing device such as a mouse. However, in the Web this model is seldom used as existing web standards have evolved around representing documents [77]. *Direct manipulation* is another demanding *interaction model* that is used in e.g. drawing programs in which visual feedback to the user has to be immediate. Similarly to WIMP, this type of interaction is nearly impossible to realize without plugins with existing web technologies using major web browsers as user agents. [95]

However, the needs of the above-mentioned interaction models in web applications have been identified and are being addressed by the evolution of the Hypertext Markup Language [98] which proposes various improvements. In addition, W3C Web Accessibility Initiative [92] aims to make more advanced features of dynamic content and Rich Internet Applications accessible. Such improvements in web technologies are further discussed in Sections 3.5 and 3.6.

Web applications that are build on top of the same web technologies can vary a great deal in how the roles and workload is split across the server-side and the client-side. As a simplification one could state that the more the client-side is responsible for running the functional algorithms the more interactive the application is. Honkala [95, p. 12] proposes the following taxonomy for web applications based on their interactivity:

**Information retrieval** Characteristics of this category are web applications with low interactivity requirements. Typical use cases include user browsing and searching information with simple search terms.

**Information manipulation** Medium interactivity such as editing predefined parts of the information by the user is analogous to adding items to or removing items from a shopping cart in an e-commerce application.

**Information authoring** Highest interactivity requirements are set by the applications requiring information authoring using specific applications. For example, word processor, spreadsheet and drawing applications fall into this category.

Out of the above-mentioned categories, existing web technologies are mainly designed for implementation of web applications belonging to *information retrieval* and *information manipulation* classes of user interaction. *Information authoring* is feasible to be implemented in desktop browsers with extensive use of client-side scripting and may require the use of proprietary plugins[3]. Additional challenges related to mobile devices are limited input and output methods, such as small keyboards and screens which complicate the implementation of *information manipulation* and *information authoring* applications.

In spite of all, the web technologies have been successfully used for implementing UIs which have previously required the use of GUI toolkits built in lower-level languages such as Java. Examples include Joost [165], which is a highly interactive desktop client for distributing TV shows, and a phonebook mashup application for the S60 mobile phone platform developed by Nokia. Both of these provide integrated access to on-device functionality and to web-based services [88].

## 2.3 Common Non-functional Requirements

This section discusses common non-functional requirements for web applications in mobile context, that is requirements which specify system qualities instead of specific behavior. Such generic quality characteristics for a software are defined in ISO 9126 standard [181]. In this section only the most relevant non-functional requirements in the domain of the Web are elaborated.

[3]For example, the de facto way of embedding video to the Web is to use proprietary Adobe Flash browser plugin. [22]

### 2.3.1 Usability

International Organization for Standardization (ISO) [180, part 11] defines the usability as the "extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use". In practice this means that user can find an element in a user interface problematic in many ways: the system may be difficult to learn, it may be slow, cause errors or be plainly unpleasant.

Usability challenges emerged from the start when Internet services started to migrate to mobile devices in the beginning of the millennium aiming to make ubiquitous information access any time anywhere reality [117]. Various usability studies have been conducted especially dealing with mobile applications [58, 130]. Another area of interest has been the web content designed for Wireless Application Protocol (WAP) [33, 117]. There have also been various usability studies in the area of mobile browsing [170, 171, 172]. The prevailing view in the above-mentioned studies has been that high level of user satisfaction is critical to the success of applications designed for mobile devices. The poor usability of the Web and web applications in particular has been a known issue since the emergence of web technologies [147].

### 2.3.2 Accessibility

There exists varying interpretations of the relationship between accessibility and usability. International Organization for Standardization (ISO) specifies accessibility as the usability extended at the largest possible number of users. [182]. W3C states that the content is accessible when it "may be used by someone with a disability" [32]. This clearly implies that something that is accessible may not be usable and vice versa. [13]

In the context of the Web, web accessibility means that the potential audience is maximized. In a wider sense it also measures "resistance to external or temporary handicaps, such as noisy environments or bad lighting" [16]. Various guidelines which incorporate the knowledge of how people perceive, understand, navigate and interact with the Web have been published. Adherence to guidelines will make the Web more available to users despite of the device used. An additional benefit is that accessible content will make information searching easier for both machines and humans [90]. Moreover, many countries have legislation which address the accessibility issues. [32, 91]

## 2.4 Guidelines for Developers

Next, major guidelines relevant to web developers of web applications are reviewed. This includes usability and accessibility as well as design guidelines and best practices which partly draw from usability and accessibility guidelines but are more concrete. In addition, typical interaction design patterns applied to web applications are discussed.

### 2.4.1 Usability Guidelines

There exists a great body of knowledge discussing the usability of *Human-Computer Interfaces* (HCI) of computer-based systems. This knowledge is commonly referred to as *usability guidelines.* The usability guidelines became popular during the 1980s when computer systems were introduced to people in the work place with no special expertise. Since, guidelines have been developed progressively and recently there has been general interest toward the usability of web applications due to the proliferation of the Web as a platform for running applications.

Nielsen is one of the most renowned web usability experts. He has released various usability guidelines related to the Web [89, 147] and identifies various usability challenges in the latest development in web applications. In a study of 46 Flash-based[4] web applications Nielsen concludes that the usability requirements for web applications are far stricter than they ever were for traditional software [148]. Nielsen criticizes the Ajax technique, which is popular among recent web applications. It allows a web page to interact with the web server on the background using standard web technologies for better interactivity. He argues that the main usability problem related to the Ajax approach is the breaking of the page-centric model of the Web, which renders the navigation controls of the web browser useless [149]. However, solutions to this problem have recently been developed and are being standardized in W3C in [98].

### 2.4.2 Web Accessibility Guidelines

W3C Web Accessibility Initiative (WAI) is the body developing web accessibility guidelines for the Web. *Web Content Accessibility Guidelines* (WCAG)

[4]Adobe Flash is a proprietary extension to web browser which is commonly used to create web applications requiring rich interaction capabilities.

[32, 27] explain how to make web content accessible, *User Agent Accessibility Guidelines* (UAAG) [114] address web browsers, other user agents and clarifies how to make them accessible. *Authoring Tool Accessibility Guidelines* (ATAG) [187] provide similar guidance for authoring tool vendors. WAI guidelines are compatible with the core web technologies that are presented in Chapter 3. [91]

*Graceful degradation*, which enhances accessibility of the Web, is an important principle in web design. It is applied to web content which is meant to be consumed by user agents of varying capabilities. The approach advocates the use of best practices to avoid incompatibility issues stemming from web browser inconsistencies enabling the web content to be fault-tolerant. For example, in a case of a failure (e.g. a lack of support in the browser for certain web technology) the operating quality of an application decreases instead of a total breakdown. It includes principles such as separation of the behavior layer from the structure and presentation layers. A reversed approach to graceful degradation is progressive enhancement [30] which is geared toward the lowest common denominator of browser functionality. [131]

The graceful degradation approach is fairly easy to support in web content which requires little interaction. For example, a static web page is completely viewable without presentational decoration. However, the heterogeneity of the host environment, the browser, and its incompatibilities have more severe coincidences in interactive web applications which rely heavily on scripting which makes designing gracefully degrading interactive web applications tedious.

### 2.4.3 Design Guidelines and Best Practices

In the Human-Computer Interaction discipline, *guidelines* or *human interface guidelines (HIG)* consist of a collection of principles and recommendations defining the look and feel for applications to provide a level of consistency. The need for more comprehensive guidelines also for the Web is becoming more critical as web applications are adopting more complex user interfaces familiar from desktops. *Best practices* are commonly used in the fields of software engineering to define generally agreed way to implement something.

The line between a guideline and a best practice is ambiguous and the terms are commonly used interchangeable. For example, in W3C Mobile Web Best Practices [166, section 1.5] it is stated that the W3C Web Content Accessibility Guidelines "are supplementary to the [W3C] Mobile Web Best Practices, whose scope is limited to matters that have a specific mobile relevance" [166].

To clarify the situation, W3C has started working on specifying the similarities and differences of the above-mentioned documents in [34].

*GUI Guidelines* target specific operating systems and give instruction how to design consistent applications. For example *Windows Vista User Experience Guidelines* [40] are the official guidelines for the designers of Windows software and *S60 Platform Scalable UI Guideline* [48] gives guidance how to design for the S60 platform used in the majority of world's smartphones.

Guidelines for designing web content for mobile devices have been covered by the W3C, the .mobi company [55], as well as various vendors of the mobile devices. W3C Mobile Web Initiative [97] launched in 2005 has a mission to make "Web on the move" happen, that is to make mobile web access as simple as web access from a desktop device. *Scope of Mobile Web Best Practices* describes "device and access network capabilities that need to be considered because of possible technical, ergonomic or economic implications" [144]. These included capabilities related to device hardware (battery, input, memory, processing power and screen), device software (capabilities, voice and multimodality) and access network (bandwidth and connectivity). In addition, aspects such as cost, availability, usability, partial attention, social situation and physical environment are covered.

W3C has published *Mobile Web Best Practices 1.0* [166] which specifies best practices for delivering web content to mobile devices. Building upon this work, a set of guidelines [55] has been released for the services in the .mobi top-level domain. The *Mobile Web Best Practices 2.0* [6] work which in its infancy in the beginning of 2008 is scoped to focus on web applications on mobile devices. Nokia has released guidelines which target more capable web browsers found in recent high-end models [150]. Similarly, Apple has released guidelines for designing web content optimized for its iPhone [108, 109].

### 2.4.4 Web Technology Design Principles

In understanding the approaches taken in designing web technologies, it is beneficial to look at the design goals and principles in more detail. In [16, 115] the general design goals for web technologies are discussed. Relevant topics within the scope of this thesis include maintainability, minimum redundancy and device independency. *Maintainability* refers to the ability to make changes to the system. It is facilitated by a manageable size and a clear structure of a specification. *Minimum redundancy* means that the overlap between the specifications should be kept small. *Device-independency* asks for specifications that do not depend on a specific device. For example, Hypertext Markup Language (HTML) is a device independent markup language in a sense that it can be displayed on the screen or read a loud with assistive software. On the other hand, Cascading Style Sheets (CSS) which provides the presentation is somewhat device-dependent; specifying fonts only makes sense on a visual medium. [16]

HTML Design Practices Working Draft [100] published by W3C discusses set of guiding principles used to develop the latest version of the core language of the Web, the HTML 5. The principles fall into four main categories: compatibility, utility, interoperability and universal access. Related to compatibility, the principles highlight the significance of supporting existing content, meaning user agents implementing the latest specification should also be able to handle most existing content. In addition, graceful degradation approach discussed in Section 2.4.2 should be made good use of to provide a fallback solution for older, but still popular user agents with lesser capabilities. Principles related to utility suggest concentrating on solving real-world problems over defining abstract architectures and adhering to separation of concerns among others. Interoperability highlights the significance of well-defined behavior, also in case of a failure, over vague or implementation-specific behavior loose in the contemporary Web. Related to universal access accessibility and media dependency – an ability of features to work across different platforms, devices and media when possible – are emphasized. [100]

It should be noted that although specifications related to Web would be implemented precisely according to the above-mentioned principles, user-agents are still the weakest links as they should implement the specifications consistently. In retrospect, this has never been the case.

### 2.4.5 Interaction Patterns

A *design pattern* is a general repeatable solution to a commonly occurring problem in software design. Design patterns can be applied to low-level implementation details as well as to interaction design. They provide a description on how to solve a problem independent of the programming language used. Interaction patterns or user interface design patterns are repeatable solutions to usability or accessibility problems in the UI design. Laakso identifies the following high-level categories for such patterns: *search, data views, storages, selecting and manipulating objects, time, hierarchies and sets* and *save and undo* [124]. [15, 95, 133]

Mobile devices incorporate more limited input and output mechanisms which complicate implementing common interaction patterns elementary to applications requiring rich interactivity. Furthermore, such mechanisms are more diverse, for example, input mechanisms may vary from simple 4-way rockers to touch screens. For output, screens may come in different resolutions and aspect ratios. In addition, alternative modalities such as voice may be employed for output. This is contrary to typical desktops which commonly employ a standard full-size keyboard and a mouse and enough screen real estate.

## 2.5 Key Requirements

The key requirements in this section are derived from the analysis of the use cases described in Section 2.1 and from the literature discussing usability (Section 2.3.1) and accessibility (Section 2.3.2) topics, guidelines and best practices (Section 2.4) as well as interaction patterns related to user interfaces (Section 2.4.5). The key requirements are categorized to *generic requirements* comprising of software quality metrics, *interaction pattern requirements* demanded by the end-users and *user interface language requirements* which specify requirements for languages used for the implementation of client-side web application and thus facilitate developers' implementation task. Before describing the key requirements, characteristics of a typical target device are introduced to understand the constraints of a typical target device in addition to the target audience, that is both the end-users and the developers of client-side applications built on web technologies.

**Devices** Mobile devices typically encompass limited input and output methods, that is small keyboards and screens. In addition, such devices have less computational capabilities, smaller memory and slower network connectivity compared to desktop counterparts. A typical device falling into this category is a Nokia N95 [46] representing a high-end mobile device announced in 2006. It employs a standard numerical keypad and a 4-way navigation controller, 2.4" diagonal screen with 320x240 pixels, 64 MB of RAM and support for third generation mobile telephony network and WLAN. What is relevant from the software point of view, it incorporates support for core web technologies.

**End-users** The key requirements aim to capture the needs of the end-users. It is assumed that the end-users prefer familiar interaction patterns from the desktop environment and that especially the higher interactivity and integrated multimedia content are factors positively affecting the user experience.

**Developers** To capture the needs of the developers who use web technologies to build applications with rich interactivity, aspects of languages which ease the development are covered. In other words, it is assumed that authors prefer languages which facilitate building applications preferred by the end-users characterized above.

The requirements associated with target devices loosely map to *generic requirements*, end-user requirements to *interaction pattern requirements* and developer requirements to *user interface language requirements* described in the following sections.

### 2.5.1 Generic Requirements

ISO 9126 standard [181] defines a quality model and characteristics to be used as a checklist for evaluating a software product. The main attributes of the model are: functionality, reliability, usability, efficiency, maintainability and portability. Such attributes relevant within the scope of the thesis are elaborated below[5] in addition to aspect related to authoring and web integration: [181]

$R_{11}$: **Functionality** Capability of the software product to provide functions which meet stated and implied needs when the software is used under specified conditions. Functional requirements for the concept implementation are defined in Chapter 5.

$R_{12}$: **Efficiency** Optimally the perceived performance should be comparable to that of native applications. Main components contributing to efficiency are the performance of the network, hardware and the software platform. ISO 9126 defines efficiency as the capability of the software product to provide appropriate performance, relative to the amount of resources used, under stated conditions.

$R_{13}$: **Portability** The UI should make the underlying technologies transparent to the user. This requires standard interfaces to device capabilities in a platform-agnostic way. ISO 9126 defines portability as the capability of the software product to be transferred from one environment to another. Portability encompass adaptability, which is defined as

[5]In addition, usability was considered important, but conducting appropriate usability evaluation was deemed out of the scope of this thesis. The evaluation is, however, recommended for future work.

capability to be adapted for different specified environments without applying actions or means other than those provided for this purpose for the software considered. The changes may relate to network connectivity, variations in the available processing power, memory capacity, alternating output or input methods.

$R_{14}$: **Ease of authoring** Applications should be easy to author. This is facilitated by re-usable software and tools for application development, such as an integrated development environment. Declarative languages do not require programming skills and are thus considered easier to author than procedural languages [95, p. 19].

$R_{15}$: **Web integration** Integration with the existing stack of web technologies and protocols and an ability to re-use them when possible is considered a fundamental part of any successful web technology.

### 2.5.2 Interaction Pattern Requirements

Supporting popular interaction patterns familiar to the users from the desktop applications has significant benefits. First, porting of applications is facilitated. Secondly, the user's mental model related to the application interaction does not have to be altered leading to better user experience. Below is a list of interaction pattern requirements derived from previous studies by Laakso [124], Pohja et al. [157] and Honkala [95, p. 32] relevant within the scope of the thesis:

$R_{21}$: **Typical interactors** Typical user interface components such as labels, input fields, lists, buttons, icons and scroll bars with dynamic behavior, properties and events.

$R_{22}$: **Master-detail** An UI component which displays more detailed information of the selected item. Can be implemented in various ways, e.g. as an accordion or as a two-column pane.

$R_{23}$: **Paging and dialogs** Ability to display multiple views (or pages) and dialogs while retaining their state between transitions.

$R_{24}$: **Repeating and nested constructs** Navigating and editing of repeating and nested data sets. A common example of such a data set is a table.

$R_{25}$: **Copy-paste and undo-redo** Copy an item or undo the last action. The former provides an intuitive mechanism for modifying the dataset whereas the latter is beneficial for moving between the states of an application in a way somewhat analogous to the browsing history built into web browsers.

$R_{26}$: **Drag and drop** Move an item within the UI to another position typically using a pointer device.

$R_{27}$: **Filtering** Allow filtering of datasets to limit the amount of data displayed to the user.

### 2.5.3 User Interface Language Requirements

W3C Working Draft *Compound Document by Reference Use Cases and Requirements* defines requirements for rich multimedia content in [4, section 1.2]. The following requirements derived from the above-mentioned requirements and research related to user interface markup languages done in [95, 175] facilitate the development of such applications and the task of a developer.

$R_{31}$: **Graphically rich content** Support graphically rich content such as basic primitives (e.g. lines, rectangles, images, text), advanced graphics (2D and 3D) and advanced graphic effects transformations (e.g. opacity, coloring, motion blur, highlights).

$R_{32}$: **Exact control of the presentation** Content author/provider has exact control of the presentation, including e.g. fonts, layout and color.

$R_{33}$: **Layout and content adaptation** Layout can be based upon device characteristics – screen size, color depth, resolution or orientation. A compliant version of the mixed content is delivered according to user agent capabilities (e.g. only textual contents of the interactive content).

$R_{34}$: **Navigation** Support for navigation (forward/backwards tab, up/-down/left/right, accesskey, pointer, voice) and graphical menus which support interactivity and animations.

$R_{35}$: **Interactive graphical menu system** Support for graphical menu systems where items have an animated action when focused on.

$R_{36}$: **Customizable presentation** Presentation can be customized and personalized across an application.

## 2.6 Summary

The assessment of common non-functional aspects of the Web – usability and accessibility – revealed problems which are especially significant in mobile context. Luckily, solutions which aim to tackle the identified problems are being developed. The review of web developer guidelines for authoring web content indicated that there exists two somewhat orthogonal approaches. The guidelines published by W3C and .mobi target low-end mobile devices, whereas guidelines published by the device vendors were mainly for high-end devices which incorporate more capable web browsers. Surprisingly, little focus was put on web applications as only one guideline was geared towards web applications [109, p. 15] and W3C was just initiating related work [6] as of 2008. This implies that the web applications are considered new phenomena still in the innovation phase, especially related to mobile devices with their unique characteristics.

The key requirements for web applications defined in this chapter are based on the literature study. The generic requirements incorporate established non-functional requirements for software. Usability and accessibility aspects were discussed but not included in the key requirements as their evaluation would have been out of the scope of this thesis. The interaction pattern requirements aggregate the popular interaction patterns elementary to any application irrespective of the technologies and platforms used. Finally, user interface language requirements outline requirements for languages used for implementing user interfaces with a specific focus on developer needs.

# Chapter 3

# Web Technologies

This chapter discusses the web architecture and core web technologies and their evolutionary versions to give reader a better understanding of the building blocks of contemporary web applications and beyond. In addition, those emerging web technologies are discussed which fulfill the key requirements not yet addressed which the existing web technologies.

Today there exist various alternatives for developing applications for mobile devices ranging from theoretically device independent paradigms such as Extensible Hypertext Markup Language (XHTML) and Java ME to device dependent paradigms such as C++ on Symbian or Windows Mobile. Device dependent approaches benefit from a deeper integration into the device capabilities, typically better performance and richer graphics capabilities compared with device independent web technologies. Regardless, developing native code for mobile platforms incur a significant cost as porting is needed to support multiple platforms whereas web technologies are designed to be device-independent.

## 3.1 The Web Architecture

Fundamentally the Web is a network of resources – documents, files or any other entities that can be identified – interconnected by links. This information space is shared by a number of information systems. Agents[1] act on the information by retrieving, creating, displaying and analyzing resources. The three bases of web architecture as defined by the W3C are: [115]

[1]In W3C terms, agent refer to people or software. [115]

**Identification** Distinct global Uniform Resource Identifiers (URIs) [10] are used to identify the distinct resources on the Web.[2]

**Interaction** Communication between agents over a network about resources involves URIs, messages and data. Agents may use a URI to retrieve a presentation of the resource, modify the state or delete the resource. The universal request/response protocol used in the Web to transfer content is the Hypertext Transfer Protocol (HTTP) [69].

**Data formats** The use of well-known, widely available open data formats such as XHTML [3, 154], CSS [18, 127] and XML [21] facilitate information sharing. Although such textual data formats have obvious advantages such as portability, interoperability and human-readability, some media such as video and audio is more sensible to be represented in more concise binary formats.

The three bases of the W3C Web Architecture [115] described above partly draw from the research done by Roy Fielding. He is one of the authors of the HTTP and introduced the Representational State Transfer (REST) architectural style for the Web in his doctoral thesis [68]. REST consists of a collection of network architecture principles which describe how resources are defined and addressed to ensure the scalability and growth of the Web. An associated term *RESTful* is like the term *object-oriented* [169]. The key constraints of REST according to Fielding are [68]:

**Client-server** The network is based on a client-server architecture consisting of multiple clients and servers. Separation of concerns is the main principle behind the client-server architecture. This is typically realized by moving all of the user interface functionality into the client to improve the scalability of the system and reduce the server load.

**Statelessness** The protocol is stateless, which as well improves scalability and simplifies the system as the server does have to recall client states. The statelessness constraint can be violated by using HTTP Cookies [123] which can be used to create a stateful session between a client and a server.

[2]An example URI `http://www.w3.org/Consortium/` comprises of:

(i) a URI scheme (`http`) defining the namespace, purpose and syntax of the URI,

(ii) a domain name (`www.w3c.org`) and

(iii) a path (`/Consortium/`).

**Cacheability** The communication is mostly cacheable to improve performance and to facilitate load balancing. The trade-off is the potential decrease in reliability if stale data is received from the cache.

**Interface uniformity** The interfaces between clients and servers are uniform, so that implementations of both the server and the client can evolve independently. For example, HTTP [69] provides an uniform interface consisting of URIs [10], methods (e.g. GET and POST), status codes, headers and content defined by content types for accessing resources.

**Layered system** The layered system allows each layer to operate independently which simplifies system components and hides the overall system complexity improving reusability. TCP/IP model[3] is the most well-known example of a layered system used in the Internet.

**Code-on-demand** Client functionality can be extended dynamically by downloading executable components, such as JavaScript code. Code-on-demand provides improved extensibility and better user-perceived performance and efficiency as the client can interact with a user locally. In addition, scalability is improved since work can be offloaded to the client. Regardless, this constraint is optional, since arbitrary code cannot be expected to work on every client in a heterogeneous network such as the Internet.

### 3.1.1 Key Concepts

In REST *orthogonality* is one of the key concepts of the Web which facilitates the evolution of the Internet protocols. For instance, the content types describing the data are defined and managed by its own entity, Internet Assigned Number Authority (IANA) [186]. New content types can be added independently of the underlying protocols, such as the HTTP, which in turn is developed by Internet Engineering Task Force (IETF) [69]. Equally, the identification mechanism utilizing URIs is independent of the interaction between agents.

*Resources* and *representations* are two fundamental concepts in the web architecture [68, 115]. W3C defines the term *resource* as "whatever [that] might

[3]TCP/IP model refers to the description of the layered computer network protocol design used in the Internet, in which the core protocols are the Transmission Control Protocol (TCP) and the Internet Protocol (IP).

be identified by a URI" [115]. An example of a resource is a web page or an image. A *representation*, that is data encoding information about the resource, consists of *metadata* describing the resource content type and message payload (*data*). Figure 3.1 illustrates the relationship between *identifier* (URI), *resource* (anything that can be identified by the URI), and *representation* (representation of the resource).

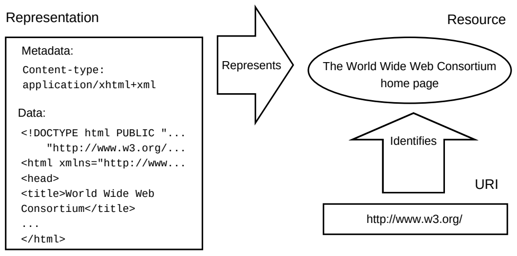


Figure 3.1: Relationship between identifier, resource, and representation. [115]

*Markup languages* are one of the foundations of the Web. The most popular markup language in the Web is the Hypertext Markup Language (HTML) [167] and increasingly the general purpose Extensible Markup Language (XML) [21] and other data formats derived from it used to share structured data.

The most popular architectural style for network-based applications – and the Web – is the *client-server architecture* which decouples clients from servers in the Internet. Clients can send requests to one or many servers and servers can accept these requests and return requested information back to the client. In the context of the Web, the client is typically a web browser, but can be any other user agent as well.

### 3.1.2 Overview of the Web Technology Stack

The web technology stack depicted in Figure 3.2 views the Web as an application built on top of the Internet. The One Web encapsulates all the building blocks above the Internet. It implies that the same information and services should be available to users irrespective of the devices they are using[4] [166, section 3.1]. The subsequent layer provides mechanisms for identifying

[4]Regardless, no golden rule is provided how to implement the One Web in practice, thus many conflicting interpretations exists.

resources using Uniform Resource Identifiers (URIs), and communication between clients and servers via Hypertext Transfer Protocol (HTTP). The next layer, Extensible Markup Language (XML) and related technologies provide the base for other application languages to build on. On top of these base layers are standard web technologies such as XHTML, CSS and JavaScript[5]. Emerging web technologies depicted with dashed lines in Figure 3.2 have not yet gained widespread adoption or are still work in progress. They are discussed further in Section 3.6.

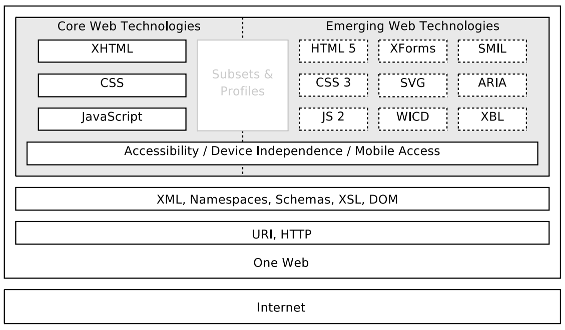


Figure 3.2: The web technology stack. Adapted from [113].

While designing for more constrained mobile devices, subsets and profiles[6] of web technologies may also be used. However, the use of subsets incur additional burden on developers who wish to publish the same content for both the more capable devices supporting full web technologies[7] and to constrained devices supporting subsets of web technologies only. Currently, constrained devices are treated as second-class citizens in the Web which is mostly built on standard web technologies. For example, JavaScript which lacks proper support in low-end mobile devices is used in 60% of the web pages as of 2007 [84, 110].

[5]To be exact, JavaScript is not standardized by W3C but Ecma International, however it has established its role as a scripting language of the Web.

[6]In standardization, a profile consists of an agreed-upon subset and interpretation of a specification.

[7]The Web content built on web technologies targeted to desktops are commonly referred to as the full web to distinguish it from the subsets targeting constrained mobile devices.

## 3.2 Base Layers of the Web Architecture

The lowest horizontal layers depicted in Figure 3.2, from bottom to top, are built on top of each other. These layers providing the base for all the other web technologies to build upon are discussed next.

### 3.2.1 Identification and Communications

As discussed in Section 3.1, a resource on the Web is identified by a Uniform Resource Identifier (URI). Commonly used term Uniform Resource Locator (URL) [11] is a subset of URI. In addition to identifying a resource, URL provides means of locating the resource by describing its primary access mechanism [10, section 1.1.3]. In practice this means that the http URI is commonly referred to as URL[8].

The information is transferred between agents using Hypertext Transfer Protocol (HTTP) [69] request/response communications protocol. The HTTP was originally designed for publishing and retrieving simple web pages, but soon it established its role as a universally used transfer protocol on the Web – mostly due to its wide support and an ability to find its way through firewalls. For example, SOAP [19] providing the basic messaging framework for Web Services stack[9] is implemented on top of the HTTP.

The HTTP defines multiple methods to act upon a resource. One can request its representation (GET) or metadata (HEAD), modify its state (POST), create or replace it (PUT), delete it (DELETE) or check methods it supports (OPTIONS). However, in practice programming libraries and user agents, such as web browsers commonly support only a subset these methods [169, section 2.3].

### 3.2.2 Extensible Markup Language

Extensible Markup Language (XML) [21] – a simplified subset of Standard Generalized Markup Language (SGML) [83] – is designed to describe data in a way that is human-legible, extensible and easy to implement. Therefore,

[8]In [10, section 1.1.3] it is recommended that "future specifications and related documentation should use the general term URI rather than the more restrictive terms URL and URN".

[9]Web Services refers to a set of computer networking protocols used to define, locate, implement, and make Web services interact with each other. Although commonly using HTTP as a transport, the REST principles described in Section 3.1 are not adhered.

it is being increasingly used as a basis for implementation of application languages. For instance, applications of XML include Extensible Hypertext Markup Language (XHTML) [154] which is an HTML [167] formulation in XML and Scalable Vector Graphics (SVG) which describes two-dimensional vector graphics. In addition, XML is used in the exchange of wide variety of structured data on the Web and elsewhere.

XML namespaces provides a mechanism for deploying multiple XML-based languages (which define their own elements and attributes) in a global environment and reduce the risk of name collision if such XML documents are combined. XML Schema definition language (XSD) express a set of rules for structure and data types for XML documents to which an XML document must conform to be considered valid. [65]

Extensible Stylesheet Language (XSL) [164] is a family of transforming languages that allow describing how XML files are formatted or transformed to other XML formats. The most relevant languages of the family in the context the thesis are XSL Transformations (XSLT) used for transforming XML documents and XML Path language (XPath) used by XSLT for addressing parts of an XML document.

To summarize, the XML has the following benefits. It is a *cross-platform* solution that can be parsed by nearly any application. XML provides a *clean namespace mechanism* allowing embedding of multiple document types in a one document. Furthermore, it has *powerful associated technologies* such as Document Object Model (DOM), discussed next.

### 3.2.3 Document Object Model

Document Object Model (DOM) is a platform and language independent object model for presenting HTML documents and XML-based formats as a collection of objects. The structure of any DOM object is a tree made up of branches and leaves. DOM provides an API that allows the document to be inspected and modified dynamically. Within a web browser the API is typically exposed to JavaScript[10].

An example of the DOM of an XHTML document (see Section 3.3.1) is depicted in Figure 3.3 where nodes represent XHTML tags such as headers (`<h1>`), paragraphs (`<p>`) or strings of text. Each node of the object tree including their properties and methods is accessed through the DOM of the

[10]It should be noted though, that implementations of the DOM can be built for any language.

document. The DOM API enables, for example, the dynamic modification of the structure of the document by modifying, removing or adding nodes by which means web applications built on standard web technologies achieve rich interaction on the client-side without need to communicate with the server. Similarly, the presentation of the elements can be manipulated and events can be added or removed. Effectively, the DOM is an interface, or a shared data structure, enabling the user agent and the scripting language such as JavaScript to communicate with each other.

W3C has established a standard for DOM and currently DOM Level 2 standard [96] is supported by major web browsers[11]. W3C has continued to refine and expand the DOM standard in the DOM Level 3 modules. [70, p. 308]

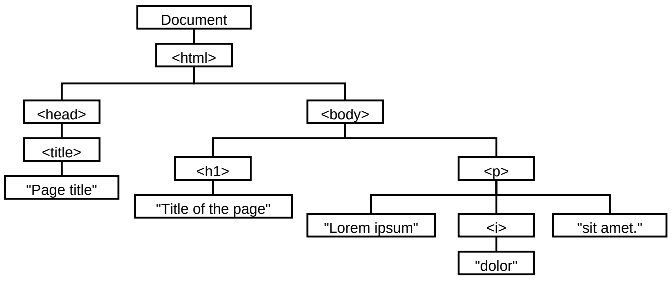


Figure 3.3: An example of the Document Object Model.

## 3.3 Core Web Technologies for Web Applications

In this section the core web technologies built on top of the base layers of the Web are discussed. These technologies are typically used for building contemporary web applications and are supported by the widest spectrum of web browsers, operating systems and platforms.

### 3.3.1 (Extensible) Hypertext Markup Language

Extensible Hypertext Markup Language (XHTML) [154] is a successor of the Hypertext Markup Language (HTML), the predominant markup language for web pages. HTML describes the structure of the textual information

[11]Internet Explorer 7 lacks support for DOM Level 2 Events [156].

within a web page. In addition, it provides means for the user to enter data to be submitted to the server using forms and defines mechanism for referencing images and other non-textual objects embedded within the web page. XHTML reformulates the syntax of HTML to an application of XML and breaks the specification into reusable and extensible modules [2].

The main motivator behind the XHTML and its modularization was the emergence of constrained devices which asked for simplified parsing[12] for complex HTML documents [154, section 1.3]. However, there have been some challenges which have undermined the successful adoption of the XHTML over HTML. Firstly, human authors tend to make errors also while writing XHTML which has stricter rules for wellformedness[13]. The claim can be easily validated by looking at the poor validity of the existing web pages [84]. Secondly, serving XHTML with `application/xhtml+xml` content type which allows using more light-weight parsers which was the main motivator, is unsupported by the Internet Explorer, the most popular web browser [93, 179].

Despite the above-mentioned shortcomings, many authors have recently adopted XHTML, perhaps without being aware of the gory details. XHTML and HTML have been successful mostly because of the right balance between the ease of learning and expressivity[14] of the language adequate for most of the needs. In a technical sense, user agents have had a major role as they have been designed to be liberal in how to interpret the markup which has lead to the explosive growth of the XHTML and HTML documents in the Web. On the other hand, this liberalism has also prevented taking full advantage of the XHTML. Regardless the differences of XHTML and HTML, they are used for the same purpose and will be generally referred to as (X)HTML in this thesis.

### 3.3.2 Cascading Style Sheets

The HTML was initially developed for scientific purposes where the content is more important than its presentation. For the purpose of styling the HTML, Cascading Style Sheets (CSS) language was developed to describe the presentation, the spatial layout, of the structured documents. The C of CSS stands for cascading which indicates that the style rules applied to an ele-

[12]Parsing transforms input text into a data structure, in the XHTML's case into a tree.

[13]A well-formed document conforms to XML's syntax rules, and is required for conforming parser to process it.

[14]Expressivity refers to how powerful a given tool or language is. [95, p. 14]

ment can come from different sources. CSS is typically applied to documents written in (X)HTML, but it can be applied to any other markup languages whose sections are clearly defined as well, such as Scalable Vector Graphics (SVG). CSS syntax is simple as it uses keywords to specify style properties such as `width` or `background-color`. Style sheets consist of selectors which map to certain elements (i.e. DOM nodes) and associated declaration block consisting of keyword value pairs which define styles for the matched elements.

For adaptation of the presentation – e.g. for mobile devices with small screens – media types of CSS provide a mechanism for accomplishing that. Each media type can include different style sheet and it is up to the user agent to decide which one to use. The most relevant media type for mobile devices is `handheld`. Unfortunately it is defined unambiguous as a device with a "typically small screen, monochrome, limited bandwidth" and thus is supported inconsistently across browsers for mobile devices [18, section 7.3].

The most recent finalized version is the CSS Level 2 [18] from 1998. CSS 2.1 – which is still not yet final – "consists of all CSS features that are implemented interoperably at the date of publication of the Recommendation" [17], that is basically a snapshot of the current state of CSS support in web browsers.

### 3.3.3 JavaScript

In the Web, JavaScript[15] is the de facto client-side scripting language that is supported by virtually any modern browser. JavaScript adds behavior layer whereas (X)HTML provides the page content and structure and CSS the presentation. Dynamic web content that uses (X)HTML, CSS, DOM and JavaScript together to create interactive effects is sometimes referred to Dynamic HTML or DHTML. The term Ajax is commonly used to describe technique which extend DHTML with a client-side capability for communicating asynchronously with the server using JavaScript.

Characteristics familiar from other dynamic languages such as Smalltalk, Self or Lisp [141, p. 3] are also visible in JavaScript. JavaScript is interpreted, i.e. it is executed command-by-command by the JavaScript interpreter[16] at runtime. This is contrary to popular languages such as Java or C/C++ which need to be compiled prior the execution. Secondly, JavaScript is dynamically

[15]To be precise, JavaScript is an implementation of the ECMAScript standard standardized by Ecma International in the ECMA-262 specification [63].

[16]JavaScript interpreter is a software component typically part of a web browser that is used to execute JavaScript code.

typed, which means variables and parameters are not explicitly declared before their use. Thirdly, interesting aspect of JavaScript is that it allows runtime modifications of the code, that is structural and behavioral aspects of the programs can be modified, by e.g. adding new functions and variables to objects on the fly. [70, p. 235]

JavaScript is dependent on the host environment it is executed in. For example, it does not have input and output constructs[17] and thus it hooks up to the Document Object Model (see Section 3.2) of the web page for this purpose. Similarly, the event-driven programming model of JavaScript depends on the web browser which generates events of the asynchronous user inputs (e.g. `click` event on mouse click), system actions (e.g. `load` event when the document is fully loaded) or network activity (e.g. `onreadystatechange` property indicating the status of the initiated HTTP connection). These events are associated with event handler callback functions implemented in JavaScript, which in turn modify the DOM giving visual feedback to the user. An example of the most recent development in the area of data formats is ECMAScript for XML (E4X) [111] which adds a native XML data type support to JavaScript. [70, p. 388]

### Asynchronous JavaScript and XML

Asynchronous JavaScript and XML, or Ajax [79], is a web development technique which facilitates creation of interactive web applications. The main component of Ajax is JavaScript and especially its `XMLHttpRequest` object which provides means to asynchronously communicate with the web server over HTTP without the need to reload the whole web page. The data received in the background can be used to modify the DOM dynamically by using the DOM API exposed to JavaScript. This allows the structure, style and behavior of the web page to be modified programmatically in JavaScript, without a need for a full page refresh. The XML part of the Ajax equation is misleading, as the `XMLHttpRequest` [118] object is able to exchange data in both plain text and XML format.

In addition to higher interactivity, another benefit of the approach is its efficiency. As only raw data coded is transferred instead of a full web page, fewer bits are transferred. Regardless, there are certain limitation and deficiencies in this approach. Due to security reasons, the HTTP communication to other than the originating domain is not allowed which makes designing

[17]For example, Java uses `java.io` package and C `stdio.h` library for input and output operations.

client-side applications accessing multiple domains tedious[18]. Furthermore, the navigation buttons of the web browser do not work as the page history is only updated when a new URI is pushed into the history stack of the web browser – a condition that does not occur automatically when using Ajax. Overall, the reliance on JavaScript can also been seen as an deficiency, especially in constrained environments, because executing program logic on the client-side may require too much computational resources.

## 3.4 Mobile-specific Web Technologies

Overly hyped Wireless Application Protocol (WAP) [71] was one of the first attempts to solve the issue of using the Internet on a mobile device in the late nineties. It did not gain traction mostly due to its approach which created incompatible protocols and data formats that were used to build operator-controlled closed services. The approach effectively decoupled the WAP from the Internet and the Web [72].

Since, more standards compliant approaches have been developed. After the release of XHTML 1.0, W3C started working on modularizing XHTML, which split the specification into logical parts such as Hypertext Module and Image Module. The latest version of the XHTML, XHTML 1.1, is based on the Modularization of XHTML [2] and adds the XHTML Ruby Annotation Module[19] [3] to the previous version, XHTML 1.0 Strict. The motivation behind the modularization was the view that a one monolithic XHTML specification did not fit well in all client platforms. Specifically mobile phones were too constrained at that time in 2001 for the implementation of a web browser supporting full XHTML. The fundamental challenge with the modularization approach taken is that the user agents complying with subsets only are incapable of displaying the existing full XHTML content.

*XHTML Basic* defines the baseline on top of which other languages complying with the XHTML Modularization are built. The target devices listed in the specification include "mobile phones, PDAs, pages, and set-top boxes" [7]. The XHTML Basic 1.1 adds support for additional modules, most notable XHTML Forms and XHTML Presentation Module.

*XHTML Mobile Profile* [73] is a superset of XHTML Basic, which was defined by WAP Forum – a consortium of mobile phone manufacturers. Its most notable divergence with the W3C approach is not complying with the

[18]Typically this is circumvented using a server-side proxy.

[19]Ruby Annotation Module is used to express text annotation used in East Asia.

module boundaries defined in [2]. Currently the specification is developed by a successor of WAP Forum, Open Mobile Alliance (OMA) [129], and it is heading towards convergence with the XHTML Basic 1.1 defined by W3C.

In addition to XHTML Mobile Profile, OMA has defined subsets of two popular web technologies, namely Wireless CSS, which is a subset of CSS Level 2 and ECMAScript Mobile Profile, subset of ECMAScript [129]. W3C introduced profiles with SVG 1.1: *SVG Tiny* and *SVG Basic*. Both are subsets of the full SVG specification and are targeted to constrained user agents – Tiny for less capable and Basic for higher-level devices. SVG Tiny is supported in Nokia's S60 Platform [151, p. 8] and in Java ME *JSR 226: Scalable Vector Graphics API* [160]. Other profiles include *XForms Basic* which describe "a minimal level of XForms processing tailored to the needs of constrained devices and environments" [62] and WICD Mobile 1.0 which "addresses the special requirements of mass-market, single-handed operated devices" [135].

The uptake of the above-mentioned subsets of the web technologies has been modest compared to the overall growth of the Web. In the next section, evolution of the core web technologies discussed in Section 3.3 is examined with a special focus on their features applicable to mobile devices.

## 3.5 Evolution of the Core Web Technologies

In Section 3.3, web technologies supported by all major web browsers were discussed. In this section evolutionary versions of these core web technologies are explored. Specifically, the focus is on features which address the requirements of mobile devices where contemporary solutions have fallen short.

### 3.5.1 HTML 5

HTML 5 is a new version of HTML 4.01 and XHTML 1.0 developed by W3C HTML working group together with the Web Hypertext Application Technology Working Group (WHATWG). It aims to address many issues of the prior specifications and allow (X)HTML to better support web applications by introducing new elements and APIs and codifying existing de facto approaches. Most of the additions to the language draw inspiration from corresponding JavaScript implementations authors have used to work around certain limitations of prior HTML versions. The XML serialization of the language called XHTML 5 is basically an update to XHTML 1.x, which adheres to XML parsing requirements such as draconian error handling. [98]

The most notable new elements introduced in HTML 5 related to web applications and specific requirements of mobile devices are:

- Additional types for `input` elements that can be used by mobile devices to provide platform native controls for e.g. selecting a date using a native calendar user interface. New self-explanatory types include `datetime`, `datetime-local`, `date`, `month`, `week`, `time`, `number`, `range`, `email` and `url` [98, section 3.1]
- A fallback mechanism for embedded content, such as unsupported image or video formats. [98, section 3.3.3.6]
- Drag and drop model which does not rely on the existence of a pointing device such as a mouse compatible with events `mousedown` and `mousemove`. [98, section 5.3]
- New more semantic elements such as `section`, `article`, `header`, `footer` and `nav` for giving meaning to fragments of a web page. This can be used by constrained devices to optimize for small screens, by for example moving the viewport automatically to the beginning of the `article` element after loading a web page. [98, section 3.8]
- The `menu` and `command` elements used for building menus can be programmatically identified and integrated to the device-native menu system. [98, section 3.18]
- The `progress` and `meter` are used to represent the completion of a task, such as downloading resources in the background. [98, section 3.12]
- The `details` element represents additional information the user can obtain on demand and can be used to implement master-detail functionality. [98, 3.18.1]

In addition to new elements introduced above, HTML 5 introduces new APIs exposed to JavaScript. The most relevant APIs that facilitate authoring web applications compared to the XHTML 1.0 are: [99, section 4]

- 2D drawing API which can be used with the new `canvas` element. [98, section 3.14.11]

- API for playing of video and audio which can be used with the new `video` and `audio` elements. [98, section 3.14.8]
- Client-side persistent storage which supports both simple key and value pairs [98, section 4.10.2] as well an SQL database interface [98, section 4.11].
- An API that enables offline web applications by providing a mechanism for storing resources identified by their URIs locally. This reduces the network traffic and allows using web applications with no Internet connection. [98, section 4.6.2]
- Editing API in combination with a new global `contenteditable` attribute allow users to edit documents and parts of documents interactively. [98, section 5]
- API for handling drag & drop operations in combination with a `draggable` attribute. [98, section 5.3.3]
- Network API to enable web applications to communicate with each other in local area networks. [98, section 6.3]
- API that exposes the history and allows pages to prevent breaking the back button. [98, section 4.7.2]
- Cross-document messaging to allow documents to communicate with each other regardless of their source domain securely. [98, section 6.4]
- Server-sent events to allow servers to dispatch DOM events into documents into new `event-source` elements. [98, section 6.2]

The main interfaces of the DOM related to HTML (see Section 4.1.2), the `HTMLDocument` and `HTMLElement`, are extended in HTML 5. The most relevant additions include method `getElementsByClassName()` for selecting elements by their class name and `innerHTML` attribute which provides an easy way to parse and serialize an HTML or XML document or an element. Furthermore, `HTMLElement` exposes convenience methods `has()`, `add()`, `remove()` and `toggle()` for manipulating element classes.

### 3.5.2 Cascading Style Sheets Level 3

Working draft of Cascading Style Sheets Level 3 specification splits the CSS specification into multiple modules. For example, *Media Queries* module expands the *media types* by specifying *media features* which allow presentation to be "tailored to a specific range of output devices without changing the content itself" [197]. Media features include properties such as `width`, `height`, `device-width`, `device-height` and `device-aspect-ratio`. Another significant addition is the module is the *Selectors* [82], which provides more powerful means for pattern matching which is the core functionality of the CSS and especially important related to interactive web applications. In addition there exist various other modules covering aspects such as fonts and presentational aspects of the CSS.

Today, the interactivity demands for web applications are approaching those of native applications. Developers have resorted to client-side scripting languages such as JavaScript in implementing rich effects achieved by scripting the style of elements via the DOM API. However, this approach is both tedious, inefficient and error-prone compared to a native CSS implementation. On this account rich interaction in the Web is typically implemented by using proprietary browser plugins, such as Flash. To address the shortcoming, extensions to the CSS Level 3 have been proposed to achieve similar functionality with a native CSS implementation in a backward compatible manner the following extension to the CSS have been proposed. *CSS Animation* [101] introduces defined animations which specify the values that CSS properties will take over a given time interval. *CSS Transforms* [102] allow elements to be transformed, that is scaled, rotated and skewed. *CSS Transitions* [103] enable implicit transitions, which describe how CSS properties can be made to change smoothly from one value to another over a given duration.

Support for CSS Level 3 Media Queries, CSS Animation, Transforms and Transitions has already been implemented in the recent versions of WebKit rendering engine (see Section 4.2.3) which is gaining momentum as the core of browsers for mobile devices. Examples of the above-mentioned CSS features are presented in Appendix A.1. [57, 197]

### 3.5.3 JavaScript 2

A new version of JavaScript language is being developed, called ECMAScript 4 (ES4) [64] or JavaScript 2 (JS2). It provides backwards compatibility to earlier versions of the JavaScript, but extends the language to support features found in other major programming languages. For example, it adds an ability to optionally use static typing which combined with JIT[20] capa-

[20]Just-in-time compilation refers to a technique for improving the runtime performance by converting the code at runtime prior to executing it.

ble JavaScript virtual machine would increase the execution performance of the JavaScript. Additionally, an alternative mechanism not requiring static typing has been developed providing significant performance gains [31].

## 3.6 Emerging Web Technologies for Web Applications

In this section, emerging web technologies, that is technologies that are under development or lack support in major web browsers, are explored. In Section 3.6.1 such technologies developed by W3C are discussed whereas in Section 3.6.2 somewhat similar proprietary approaches are reviewed. In addition, main client-side extensions and essential data interchange format are briefly discussed.

### 3.6.1 Open Web Standards

#### XForms

XForms [61] is a W3C Recommendation that aims to solve issues with the (X)HTML forms on data collection and submission by separation the purpose from the presentation. It is not a stand-alone document type and is intended to be integrated into markup languages such as XHTML.

The separation of application data from the user interface facilitates building modular software which is easier to maintain. Furthermore, most of the user interface processing associated with HTML forms is declared on the client-side in XForms which reduces latency and network bandwidth usage in addition to server load. XForms allows implementing interactive forms using its declarative markup incorporating functionality which today require scripting if plain HTML forms are used. Additional benefits of the declarative approach are simplified authoring and development of authoring tools. In addition, it also enables XForms to be more device and modality independent. XForms leverages existing web standards such as CSS, XPath and XML and adheres to the REST architectural style discussed in Section 3.1. For constrained devices there exists a more light-weight XForms Basic [62] which does not include full schema support. [95, p. 21]

### Scalable Vector Graphics

Scalable Vector Graphics (SVG) is an application of XML for describing two dimensional graphics comprising of vector and raster graphics and text. The SVG can be used to implement temporal interaction defined and triggered either via its declarative markup or via scripting, using e.g. JavaScript. Main benefits in using vector graphics over raster graphics are adaptability to various screen sizes, zooming capabilities, animation, searchability, efficiency and bindings to other existing web technologies such as DOM. However, SVG is no alternative for immediate mode rendering such as 2D drawing API, the *canvas*, defined in HTML 5. It is obvious that SVG as a DOM oriented technique has inherently bigger performance and memory overhead. Regardless, SVG has proven to be a versatile language, for example it has been used for implementing a complete windowing system running within a web browser [143]. [67]

### Web Integration Compound Document

A Web Integration Compound Document (WICD) is an on-going work within W3C which ties the core presentational web technologies together. The WICD is targeted as a standard way to build web applications which require rich interaction combining technologies such as (X)HTML, CSS, SVG and JavaScript. Furthermore, the WICD specification defines separate profiles for desktops and mobile devices. [134]

### Synchronized Multimedia Integration Language

Synchronized Multimedia Integration Language, SMIL, is an XML-based language for creating interactive multimedia presentations which may integrate various media types including audio and video. The main use case for SMIL is to define the temporal behavior, associate links and describe layout of such content. A module of latest version of SMIL, the SMIL Timesheets [190], can be seen as a temporal counterpart of CSS. Timesheets approach keeps content and styling separate, does not break existing functionality as it uses its own XML namespace and leverages standard selectors as hooks to integrate timing into a wide range of XML-languages such as XHTML and SVG. There exists a Timesheets JavaScript implementation [189] for backward compatibility with the existing browsers. [24]

#### Accessible Rich Internet Applications Suite

The most prominent solution for implementing accessible interactive web applications is proposed by W3C Web Accessibility Initiative (WAI) in *Accessible Rich Internet Applications Suite* (ARIA). It defines how to make more advanced features of dynamic content and Rich Internet Applications accessible. The accessibility improvements are implemented by two mechanisms. First, an XHTML module supporting role classification of elements [14] is leveraged. It can be used to define information on what the object is. Second, a syntax for adding accessible state information and author settable properties for XML is defined in *States and Properties* module [174]. Such information may contain e.g. information of the sort order of a dynamic dataset within a table. These accessibility features specified in ARIA can be implemented in compliance with the existing web standards supporting XML Namespaces as they add only semantics on top of the existing markup interpreted by the user agents supporting ARIA[21].

#### XML Binding Language

The XML Binding Language (XBL) is a declarative language for binding an arbitrary XML element to a binding element. This provides a mechanism for defining the behavior or presentation of the arbitrary element separately in the binding element. This enables better separation of concerns than it is possible with the core web technologies discussed in Section 3.3. The concept is similar to CSS and attaching event listeners programmatically via JavaScript, but adds an extra layer of abstraction with intent to simplify the development. It should be noted that XBL will not replace existing core web technologies such as CSS and JavaScript but enhances them. Initial version of the language was proprietary to Mozilla used as part of their Gecko rendering engine, but the next version XBL 2.0 [199] is being standardized in the W3C.

### 3.6.2 Proprietary Application and UI Markup Languages

W3C Web Application Formats Working Group did have plans to publish a specification for declarative format for applications and user interfaces [112]. However, the work on the specification was stopped main technical reason being that the use cases and requirements [193] for such a declarative format

[21]Out of the major web browsers only Mozilla Firefox currently supports ARIA. [74]

can be addressed by existing open standards (see Section 3.3) and by open standards in progress (see Sections 3.5 and 3.6.1) [8]. Next, the most prominent existing proprietary application and UI markup languages are reviewed, which were meant to provide the base for the above-mentioned W3C work.

#### Extensible Application Markup Language

Extensible Application Markup Language (XAML) is a declarative XML-based user interface markup language for defining UI elements developed by Microsoft. In order for XAML to be supported in web browsers, a proprietary Microsoft Silverlight (see Section 4.2.4) plugin must be installed to the browser. From developer's perspective maintaining XAML code practically requires an IDE[22] as the syntax does not separate content, structure, style, behavior or data bindings. XAML defines its own syntax for vector graphics in addition to styling language instead of leveraging the standard web technologies SVG and CSS. The key strength of XAML can be considered to be its supports for some functionality that is currently missing from established open web standards specified by W3C, such as automatic declarative transition animations, gradients, filters and styling of form controls. However, HTML 5 and upcoming CSS versions are tackling most of the above-mentioned issues, in addition to existing SVG standard which is only impeded by limited user agent support.

#### XML User Interface Language

XML User Interface Language (XUL) is an XML markup language developed by the Mozilla project that is used in its cross-platform applications such as Firefox web browser to define the layout of the application. It defines desktop-centric UI components such as windows, panels and dialogs. XUL leverages existing web standards such as CSS, JavaScript and DOM. Additionally, behavior of the XUL components can be described using XML Binding Language (XBL) discussed in Section 3.6.1.

### 3.6.3 Client-side Extensions

This section takes a look at client-side extensions which facilitate the development of client-side web applications.

[22]Integrated development environment (IDE) is an application that facilitate software development.

#### Client-side Databases

Emergence of more advanced web applications has underlined the lack of proper mechanism for storing data on the client-side persistently. The standard approach of HTTP Cookies [123] is insufficient for storing larger amounts of data.

Google Gears [86] is an open source browser extension which lets developers create web applications which can store data within the browser enabling applications to run offline. This is implemented by running a local server which cache and serve application resources and provides a database used to store and access data from within the browser. Similar functionality is being codified in the HTML 5 [98] which specifies APIs for client-side persistent storage and a mechanism for storing resources locally as discussed in Section 3.5.1. These parts of the specification have been partially implemented by the latest versions of Firefox and WebKit-based browsers. [98]

#### Client-side Cross-site Requests

W3C Working Draft *Access Control for Cross-site Requests* [119] is specifying a mechanism to enable client-side cross-site requests, main use case being the use of `XMLHttpRequest` object. In practice this is implemented either via HTTP headers [69, section 14] or via XML processing instructions [20, section 2.6]. For example, a resource at `my-domain.com` is able access another resource at `example.org` if the resource served at `example.org` is served with given HTTP headers or contains given XML processing instructions. Some web runtime environments such as S60 Web Runtime discussed in Section 4.2.5 allow accessing multiple domains without the need to use the above-mentioned mechanism as they install source resources to the local filesystem.

### 3.6.4 Essential Data Interchange Formats

#### Feeds Formats

XML-based formats used to publish frequently updated content, such as blog posts, on the Web are referred to as feeds. The feeds are used by feed reader applications which display their content to the user. The terms feed, web feed, RSS feed and XML feed are commonly used interchangeably although this is not technically valid as their schemas differ from each other [155]. Information embodied within feeds is more semantical compared to web pages containing the same information. For example, all the feed formats define elementary properties such as publishing time, title and description explicitly. This characteristic of feed formats makes them especially applicable for conveying information in a more conservative manner in terms of computational, memory and network bandwidth requirements.

The most popular feed formats include various versions of RSS[23] [185] and Atom Syndication Format [152]. Despite various improvements over RSS, the Atom Syndication Format has not yet gained as widespread adoption as RSS.

### Outline Processor Markup Language

To tackle portability issues arising from managing multiple feeds across various devices and software, an XML-based aggregate format, Outline Processor Markup Language (OPML) [196], has been developed. The specification defines an outline as an ordered list with a hierarchy of arbitrary elements and therefore is applicable for various use cases. OPML is commonly used to exchange lists, or aggregates, of web feeds between reader applications. The specification itself is fairly simple and has not yet been formally standardized by a standardization body.

### JavaScript Object Notation

The JavaScript Object Notation is a format that is typically used for interchanging data between the client and the server in web applications leveraging Ajax technique. Although it is based on JavaScript language, the data format itself is language-independent and serves as a light-weight alternative to XML-based formats. [56]

## 3.7 Summary

This chapter introduced the web architecture and core web technologies used as building blocks of contemporary client-side web applications. In addition, evolutionary versions of the core web technologies were discussed as well as emerging web technologies targeted specifically to facilitate web application development. The chapter was concluded with a comparison between the

[23]Depending on the version, RSS may stand for Really Simple Syndication (RSS 2.0), RDF Site Summary (RSS 1.0 and RSS 0.90) or Rich Site Summary (RSS 0.91).

current baseline and the evolutionary and emerging standard web technologies depicted in Table 3.1. The results imply that the existing core web technologies do not sufficiently fulfill the needs of modern web applications. This claim can be validated by looking at the large number of emerging web technologies which aim to provide more robust solutions to the identified problems. Currently, the lack of user agent support is hindering the adoption of these more appropriate technologies over widely supported baseline in web application development.

Table 3.1: Comparison between current baseline and emerging standard web technologies. Baseline web technologies refer to technologies supported by major browsers listed in Appendix B.1.

| | Standard web technologies | |
|---|---|---|
| Scope | Current baseline | Emerging |
| Structure | HTML 4, XHTML 1.x | (X)HTML 5, XBL |
| Layout and styling | CSS Level 2 | CSS Level 3, XBL |
| User interaction and data model | JavaScript | JavaScript 2, XForms, XBL |
| Timing | JavaScript | JavaScript 2, SMIL Timesheets |
| Data Interchange | RSS | Atom, OPML, JSON |
| Vector graphics | - | SVG |
| Accessibility | - | ARIA |
| Integration | - | WICD |

# Chapter 4

# Web Applications and Platforms

In this chapter, web applications are discussed and classified according to their interactivity and architectural approaches. Second, major underlying platforms for web applications are reviewed.

## 4.1 Web Applications

In the context of this thesis, a web application is defined as an application built on standard web technologies (see Section 3.3) using HTTP for communication over the Internet. Typically, a web browser or similar software available on multiple platforms supporting standard web technologies is used to run web applications. Despite the popularity of web applications, there are still some challenges which undermine the utility of web applications, namely:

- lack of highly interactive rich graphics capabilities,
- no standard way to implement offline functionality when no network connectivity is available and
- limited access to device capabilities.

Mainly on this account, new web application platforms have emerged that extend the browser to address the above-mentioned shortcomings. In this thesis, this class of user agents is referred to as *web runtime environments* which are discussed more in Section 4.2.4. From the constrained mobile device point of view web runtime environments have potential to address many limitations discussed above. Typically they integrate more deeply with the underlying platform which provides richer graphics capabilities. In addition, the integration may open access to other platform capabilities such as persistent storage for offline support and location information valuable for context-aware web applications among other functionality.

### 4.1.1 Web Application Architectures

#### Multi-tier Architecture

Traditional multi-tier architecture refers to an architecture where the application is executed by multiple distinct software agents. For example, traditionally web applications have relied on the server for executing application logic. A common approach has been to split the application into orthogonal parts, such as those proposed by the three-tier architecture comprising of the presentation tier, the logic tier and the data tier. In this model the *presentational tier* provides the UI to the user and is implemented by serving static HTML and CSS to the web browser. The *logic tier* generates web content dynamically on the server-side using server-side languages such as Java, PHP or Python. The *data tier* takes care of storing the application data persistently to the server-side, typically into a database of some sort. Typical of this traditional three tier model is that the functional algorithms[1] reside completely on the server-side.

What is different from the traditional approach described above, contemporary web browsers can also execute program code, typically JavaScript, on-demand in the browser. In addition, various client-side persistence mechanisms have been recently developed and platform-specific APIs are being exposed to the browser context. The motivation for such development has been to enhance the user-perceived performance and enable more efficient use of the network and server resources by shifting the responsibilities of the server, such as executing heavy computational tasks, to the client. In addition, the intent in exposing platform APIs to web applications has been to provide similar functionality familiar from native applications – such as access to location via GPS – also to web applications.

Taking the above-mentioned advantages into question, the trend of using the client-side as a runtime platform is understandable. However, it poses greater

[1]Functional algorithms handle the information exchange between the UI and the underlying layers of an application above the database – sometimes referred to as business logic.

demands on the standards compliance of user agents. Secondly, it assumes that computational resources on the client-side are available. Thirdly, there are certain security aspects in running applications locally using web technologies that need to be understood. Overall, the coordination between the client and the server may become more complex if the application logic is split on both the sides as the developers need to deal with both the server-side development as well as the client-side web technologies, which most likely are implemented in different languages.

### Model-View-Controller Architecture

The Model-View-Controller (MVC) is a commonly used architecture in GUI applications. According to Fowler [78], the MVC can be interpreted in many different ways. For example the interpretations vary for GUI applications and web applications requiring client-server communications.

In traditional thin client web applications, *model* is the domain model stored on the server, the *view* the generated content sent to the browser, and the *controller*, a server side component defining the workflow of the application. In desktop GUI applications, the components of the MVC are predominantly implemented within the client without similar division between client and server. Modern web applications may take hybrid approach which combines both the approaches as clients have become capable of running program code independent of the server. Regardless, three common components can be identified in any application implementing MVC:

**Model** Model objects which hold data and define the logic used to manipulate the data. Model objects are not directly displayed to the user. Typically, they are reusable, distributed, persistent and portable across platforms.

**View** View objects represent visible parts of the application, that is the UI or parts of it.

**Controller** Controller is a mediator object between the model and the view objects. It communicates data between the model objects and the view objects. Application specific tasks such as user input processing and loading application configuration data are also responsibilities of the controller.

### 4.1.2 Web Application Programming Interfaces

Application Programming Interface (API) is a software interface that enables one program to use facilities provided by another. In this section various APIs related to client-side web application development are discussed.

#### Standard Client-side Web APIs

According to W3C, APIs related to the client-side web technologies cover "programming interfaces for client-side development, including network requests, timed events and platform interaction" [60]. Next, an overview of well supported standard and de facto APIs used for implementing client-side applications is given.

**DOM Core and HTML** DOM Level 2 comprises of modules which break the functionality into language neutral interfaces[2]. The DOM Core [125] interfaces provide extensive set of methods for creating, manipulating and gathering information about elements and attributes. The DOM HTML [183] interfaces extend the Core to add functionality specific to HTML documents. Interfaces defined in HTML module are: an `HTMLDocument`, the root element in the HTML that holds the entire content and `HTMLElement` interface which is the base for all HTML element interfaces providing convenience methods which try to make the API easier for web developers. However, this model which forms the base for web applications has also been criticized for being overly complex and inefficient [141].

**DOM Events** DOM Level 2 Event Model [156] specifies an event system for registering event handlers, describing event flow through the DOM and providing contextual information for each event. The events are broken out by event modules to e.g. user interface, mouse and HTML events. The API is supported by other major browsers except Internet Explorer which implements its own more restricted model [70, p. 409]. [156, section 1.3]

**Client Interface** An intermediate layer between the JavaScript and the DOM is commonly called the Browser Object Model or BOM. Its task is to manage browser windows and enable communication between them. The centerpiece of this functionality is the `window` object, a de facto

[2]Important modules include *Core* [125], *Events* [156], *Style* [195] and *HTML* [183].

standard that is being formally standardized in W3C [59]. It provides a global object needed for JavaScript to function, represents the browser window to the user, grants access to loaded documents and provides a mechanism for primitive timesharing which can be used for rudimentary multitasking among other miscellaneous functionality. [122]

**HTTP Functionality** The `XMLHttpRequest` object implements an interface allowing scripts to perform HTTP client functionality, for example to submit form data or load data from a server. [118]

### APIs for Client-side Access to Device Capabilities

Currently, there exists no standard for exposing device or platform capabilities to applications running in web browsers. Instead, various incompatible approaches for exposing such functionality to web applications exits as it will be discussed in Section 4.2.4. The benefits of such a tighter integration are obvious: the approach allows authors to use the familiar web technologies to leverage functionality provided by the underlying platform and the device.

Within W3C there is a work on-going to address the interoperability issues specific to user-installable web application called widgets (discussed in Section 4.1.3) by standardizing selected client-side APIs including, for example, a mechanism for storing data persistently on the client [120, Section 5.2]. Also HTML 5 discussed in Section 3.5.1 is specifying new client-side APIs. Furthermore, W3C Ubiquitous Web Applications Working Group is working on specification for language neutral APIs that provide web applications access to dynamic properties representing device capabilities in [191]. In addition, a model for the device characteristics relevant for the Web is being specified in [126]. To summarize, currently the foremost risk is the fragmentation of the above-mentioned APIs which would hinder the uptake of more advanced web applications leveraging device capabilities.

### Open Service APIs

Growing trend in web applications is the use of data from multiple sources. For example, combining point-of-interests from one source to cartographical data from another source forms a hybrid web application that is commonly referred to as a mashup. Fundamental building blocks of mashups are open service APIs which implement a simple interface that transmits domain-specific data over HTTP. Two most popular options are the REST architectural style (see Section 3.1) without an additional messaging layer or SOAP [19] messaging framework facilitating the exchange of XML-based messages. Recently, many popular providers of open service APIs have been adding support for JSON [56] as an alternative data interchange format in addition to XML.

### 4.1.3 Classification of Web Applications

In this thesis, web applications are divided into the following three categories based on their interactivity and architectural approach: *thin client web applications*, *interactive web applications* and *rich client web applications*. Interaction models and associated components of web applications belonging to each of these categories are depicted in Figure 4.1. The server component is simplified and consists of a mandatory part, the HTTP server serving the resources to the clients. Optional components encompass the UI and application logic.

Figure 4.1 suggests that rich client web applications depicted on the right may execute all logic on the client-side and utilize the server only as a data source via open service API using HTTP as the communications protocol. The arrows in the figure depict HTTP communications where solid lines are synchronous typically requiring user actions and dotted lines asynchronous communications using the `XMLHttpRequest` object performed in the background and controlled by logic implemented in JavaScript. On the client-side, static (X)HTML and CSS are the core components for thin client web applications which are constructed into a static DOM tree representation of the document. In interactive web applications JavaScript is used to implement UI logic, such as changing component visibility. In rich client web applications, client-side it is responsible for overall application and UI logic. The database components in the figure illustrate that also client-side may be used for storing significant amount of data persistently using mechanism such as those discussed in Section 3.6.3. Similarly, Platform APIs may be exposed to the rich client web applications to provide hooks to the underlying platform.

It should be noted, that the figure is just a one interpretation of the state of web applications and in practice associated terms and techniques are used ambiguously. Secondly, the classes discussed below are not mutually exclusive, rather they coexist on the Web today. In addition, from the viewpoint of software engineering principles, there is a need to apply more rigorous software engineering practices to the development of web applications [142, p. 7]. This is especially visible in rich client and interactive web applications classes where plethora of approaches exists. This makes defining a detailed yet generalized interaction model an impractical task.

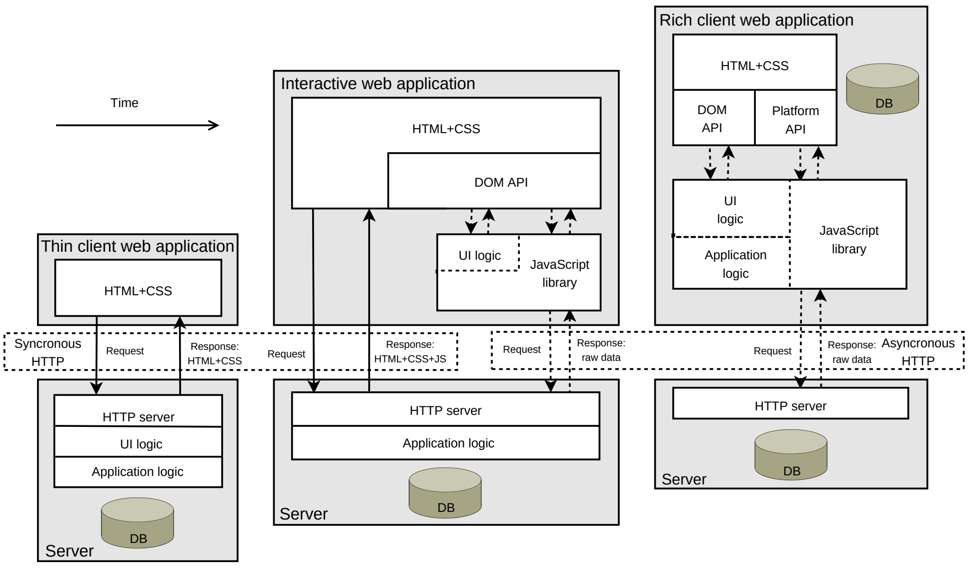


Figure 4.1: Thin client application (left), interactive web application (center) and rich client web application (right) interaction models. Adapted from [95].

## Thin Client Web Applications

Thin client web applications represent simple or classic type of web applications which use the web browser only for rendering the content to the screen and communicating with the server over HTTP. Truly they are just web pages in which interactions other than scrolling the page or filling in forms require the browser to initiate an HTTP request to fetch a new web page from the server. This implies that a lot of redundant data is transferred on each page refresh assuming most of the page content remains unchanged. Thin client model leads to poor interactivity and high network usage in term of transferred data. The approach typically fulfills the requirements of applications falling into the *information retrieval* category.

## Interactive Web Applications

The main difference with interactive and thin client web applications – or simply put just static web pages – is that an interactive web application thrives to be more interactive and typically is process-focused rather than content-driven. In this thesis the term interactive web application refers to a class of web applications which utilize the Ajax technique (see Section 3.3.3) allowing asynchronous use of HTTP via the JavaScript `XMLHttpRequest` [118] object using standard web browser with no extensions as a user agent. The main motivator in using the Ajax technique is to enhance the interactivity stemming from the fact that an application utilizing Ajax does not have to do full page refreshes to change the view after the initial page load.

In practice, authors typically mix the thin client approach with some asynchronous HTTP and client-side DOM manipulations provided by some sort of a JavaScript library (see Section 4.2.3). The libraries are referenced from the (X)HTML and loaded similarly to any other resource implying they are also saved to the browser's cache as depicted in Figure 4.1. Subsequent requests typically return only parts of the overall page payload. The cumulative data transmitted is usually lower than in a similar application based on the thin client interaction model as part of the interaction within the browser happens without a round-trip to the server. This reduces the perceived latency beyond the initial page load as less data is transmitted from the server to the client.

This approach has made it feasible to design responsive desktop style user interfaces for web applications running in a web browser and brought the user experience close to that of typical desktop applications. Interactive web application model is especially suitable for applications in *information manipulation* category which require higher interactivity than the thin client approach can offer.

### Rich Client Web Applications

Rich client web applications, or alternatively Rich Internet Applications (RIAs), refer to an emerging category of applications built predominantly using web technologies embodying close the features and functionality that of traditional desktop applications. Compared to web applications belonging to above-mentioned categories, RIAs tend to offer richer functionality, more responsive UI and be more efficient in term of network usage.

From the technical point of view RIAs share one common characteristic by introducing an intermediate component between the web browser and the underlying platform residing on the client-side. This intermediate engine component residing on the client-side may be downloaded once the application is started, is provided as an additional stand-alone installation or is pre-installed. The intermediate engine extending or replacing parts of the browser is typically responsible for e.g. rendering parts of the user interface and exposing access to selected platform capabilities. Consequently, RIAs tend to do inherently more processing on the client and require less client-server communications than their traditional web application counterparts. Typical extensions to standard web technologies leveraged by RIAs include client-side persistent storage mechanism and access to platform capabilities, such as utilizing native UI components. The former enables handling bulk of the data processing on the client-side and de-coupling the applications from the server whereas the latter allows more seamless integration to the underlying platform and its capabilities.

Currently, there exists no widely agreed definition or architecture for RIAs and on this account various platforms incompatible with each other have been lately introduced. The most popular such platforms are discussed in Section 4.2.4.

### Widgets

A specific type of rich client web applications is referred to as widgets[3] executed within a *widget engine*. In this section, the term widget refers to "a class of client-side web application for displaying and/or updating local or remote data, packaged in a way to allow a single download and installation on a client machine or device" [120]. Furthermore, within the scope are only widgets implemented in standard web technologies. The widget engine, is either directly built on, or provide similar functionality to, a common web browser [26, section 1.1] but without browser chrome[4].

The distribution of widgets is handled similarly to native user-installable applications. Herein, they differ from other types of web applications which are entirely delivered on-demand or do not require to be installed prior use. Furthermore, widgets act more like traditional desktop applications and they do not necessarily require network access to function. Widgets are typically conservative in their use of network resources as the application and UI logic can be persistently installed to the device during the initial installation. It is also common that a platform specific APIs are provided to allow widgets to interact with the host platform. For communications, widget engines use

[3]The term widget is overloaded, and it is ambiguously used to cover desktop widgets run on desktops, web widgets embedded in web pages and mobile widgets run on a mobile device. Widget may also refer to GUI widget which is an interface element of a UI.

[4]The browser chrome refers to browser UI elements such as borders of a web browser window, window frames, menus, toolbars and scroll bars.

HTTP, typically via `XMLHttpRequest` object.

Currently, there exist separate widget engines for Nokia S60 Web Runtime Widgets, Yahoo! Widgets, Google Gadgets, Microsoft Gadgets, Apple Dashboard Widgets and Opera Widgets. Although widgets are mostly built on standard web technologies such as (X)HTML, CSS and JavaScript, the vendors have introduced proprietary extensions, such as XML formats used for metadata and specific APIs used to access platform specific features. Consequently, the main shortcoming of widgets is currently the lack of established standards, due to which widgets do not interoperate across platforms. Regardless, it has been demonstrated that it is possible to port a widget to other platform with a little effort if the building blocks of the widgets are similar [116]. Widget specifications such as Apple Dashboard Reference [106], Opera Widgets Specification 1.0 [177] and Nokia Web Runtime Widgets API Reference [52] define many uniform components, which facilitates porting widgets from a platform to another [116].

Standardization of widgets is under way in W3C and Widgets 1.0 Working Draft [120] aims to standardize the core features of widgets. Those include a packaging format to provide an interoperable way to encapsulate and distribute widgets, an XML-based configuration format and processing model for widget metadata, and a model that allows a user-agent to automatically start a widget. In addition an HTTP-based model for version control and a set of ECMAScript implementable DOM APIs and events is specified. Furthermore, a model that allows a widget to be digitally signed besides a security model is defined. Finally, the specification defines means for web browsers to automatically discover widgets from within an HTML document in addition to accessibility requirements for user agents. A significant finding is that the above-mentioned features do not aim to standardize access to device capabilities. [25, 26]

### 4.1.4 Feed Reader Applications for Mobile Devices

Applications that consume feeds discussed in Section 3.6.4 are commonly referred to as feed readers. Next, an example of an application belonging to this category developed in both device-dependent and device-independent languages is introduced.

#### Web Feeds

Web Feeds is the default feed reader application of the S60 platform. As a native S60 application it is implemented in C++ and its UI adheres to S60 UI Guideline [48]. Integration with the Web Browser of S60 enables it to discover and open feeds directly from within the web pages displayed in the browser. Web Feeds application allows storing feeds persistently for offline viewing but does not provide support for the OPML aggregate format.

#### Google Reader

Google Reader [87] is a feed reader implemented in web technologies running within a regular web browser. Based on the classification introduced in Section 4.1.3, it falls in between the interactive web application and rich client web application categories. For offline capability, it optionally leverages Google Gears extension discussed in Section 3.6.3. Other key features of Google Reader include ability to aggregate feeds together from multiple sources, support for OPML format and powerful search functionality. Specifically to mobile devices, there exists a more light-weight UI alternative leveraging XHTML Mobile Profile.

## 4.2 Platforms for Web Applications

A *platform* is a set of services, technologies and application programming interfaces (APIs) on which additional software and applications are constructed. Figure 4.2 depicts a simplified overview of the platforms for client-side web applications as horizontal layers. Web applications discussed in Chapter 4.1 and their connections with the platform layers are depicted as emphasized vertical boxes. *Native applications* refer to applications implemented in the language native to the operating system. The main APIs exposed to developers are illustrated with dashes lines.

*Hardware* platform refers the physical components and associated features of the mobile device including physical construction, CPU architecture and radio interfaces for network connectivity.

*Operating system* layer consists of an operating system whose main function is to manage the sharing of the resources of a mobile device and provide a base platform for middleware and other software atop.

In the context of this thesis, *middleware* refers to software between the appli-

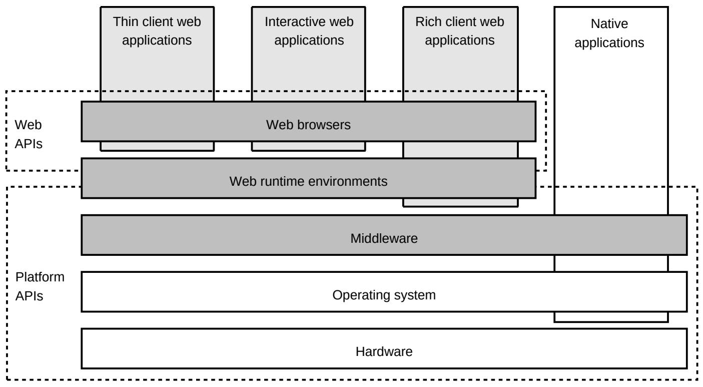


Figure 4.2: Platforms for web applications. The horizontal layers depicts platforms and vertical emphasized boxes web application categories.

cation software and the operating system. Characteristics of middleware are standard programming interfaces and protocols which provide higher-level interfaces that hide the underlying complexity and allow developers to focus on application-specific issues.

*Web runtime environments* refer to runtime environments[5] that can be seen as glue between the middleware and the web browser. Applications implemented on top of such platforms leverage web technologies but require additional component which extends or replaces parts of the web browser's functionality. [12, 66]

*Web browsers* are increasingly used as ubiquitous platforms for running applications built on standard web technologies. These highly-portable web applications provide rich interaction capabilities but suffer from mediocre performance and tight security policies preventing them from leveraging the underlying platform capabilities among other limitations.

Standard client-side *Web APIs* discussed in section 4.1.2 are the main interfaces utilized by web applications. *Platform APIs* refer to the APIs that expose operating system capabilities, such as file system access, to applications. Figure 4.2 roughly depicts the availability of APIs available to developers in various platforms.

[5]Runtime environment, or runtime, is a loosely defined term for a portable programming environment and related libraries used to execute the applications typically adhering to *write once run everywhere* idea. For example, Java Runtime Environment denotes Java Virtual Machine used to execute the code combined with standard class libraries implementing the core Java API. [140]

### 4.2.1 Operating Systems

The most popular operating systems for high-end mobile devices that support software development by third parties are Symbian OS, Windows Mobile and various offerings based on the Linux kernel.[6]

Symbian OS is a proprietary operating system designed for mobile devices. It provides the base OS on top of which additional features such as GUI and more extensive programming libraries can be build. Features of Symbian OS such as preemptive multitasking, multithreading and memory protection are familiar from modern desktop operating systems. Windows Mobile developed by Microsoft is also a proprietary operating system targeted at mobile devices. However, it is not limited to smartphones as it is run also on e.g. PDAs and media players. Different from Symbian OS, Windows Mobile includes a suite of basic applications such as e-mail client and a web browser which come with S60 platform to Symbian OS powered devices. In addition, there exists a plethora of Linux-based operating systems for mobile devices with do not yet enjoy commercial success [128].

### 4.2.2 Middleware

In this section the most popular and remarkable emerging middleware platforms for mobile devices are discussed. All of the middleware platforms reviewed primarily focus in facilitating development in their native language, be it C++ or Java. However, they are increasingly adding in support for popular web technologies and typically provide a mechanism for leveraging web technologies from within the native application development environment.

#### S60 Platform

The S60 platform is the major software platform for Symbian-based mobile devices. It accounted for 74% of all Symbian mobile devices shipped in the first half of 2007 [80]. S60 comprises of libraries and standard applications which extend Symbian OS. There exist multiple versions of S60 which provide additional elements to the platform. Figure 4.3 represents the architecture of S60 platform as of version 3.1. The main elements of the S60 are discussed below [151]:

[6]According to Gartner, Inc. in the first quarter of the 2007 the most popular operating systems in mobile devices were Symbian OS (71%) and Linux (16%). Microsoft Windows Mobile market share was below 5%. [80]

**Symbian OS Extensions** A set of capabilities which allow S60 to interact with device hardware functions.

**Open C** A subset of Portable Operating System Interface for Unix (POSIX) [104], which is an API for software compatible with Unix variants.[7]

**S60 Platform Services** A service framework comprising of capabilities for e.g. UI components, graphics, location, multimedia and communications and web-based services.

**S60 Application Services** Provides higher level services to applications such as managing contacts, calendar, messaging and browsing.

**S60 Java Technology Services** The Java ME implementation of the S60.

**S60 Web Runtime (WRT)** WRT is a runtime environment which enables running applications, or widgets, build on web technologies on the S60.

**S60 Applications** Pre-installed applications embedded within S60. The most relevant application related to web technologies is the Web Browser for S60.

**User Interface** UI libraries for developers allowing creation of custom components derived from the standard libraries.

The most relevant components of the S60 related to web technologies are the S60 Web Runtime discussed in section 4.2.4 and its enabling S60 application, the Web Browser for S60 discussed in Section 4.2.3. In addition, there is a mechanism for embedding web content within a native S60 application via Browser Control API [47] part of S60 Application Services. Web technology support in Java is discussed in the following section.

### Java ME

Java ME is a subset of the Java platform which allows developing and running programs written in Java language in constrained devices with limited memory, display and power capacity. Being cross-platform runtime environment, the main advantages of Java ME are its wide support and relatively rich feature set. It is available in some form on almost any mobile device on

[7]Open C facilitates porting applications from desktop environment to S60 by providing middleware C libraries.

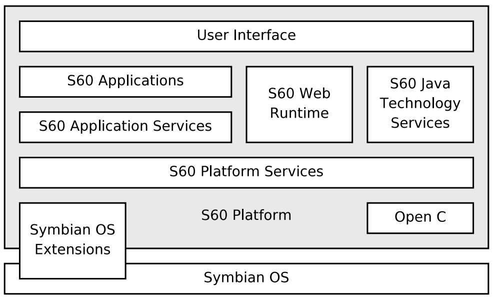


Figure 4.3: Overview of the S60 platform architecture. Adapted from [151].

the market and it allows access to selected platform capabilities such as multimedia and graphics functionality via standard APIs. However, in practice also Java ME applications distributed as user-installable MIDlet files need to be adapted for various subset of mobile devices available, making the development of Java ME applications expensive and time-consuming, especially when optional functionality is used [188]. The main architecture of Java ME consists of: [140, 139]

**System libraries and configurations** The most basic set of libraries and the virtual machine used to execute the programs.

**Profiles** A set of higher-level APIs to extend the configurations.

**Optional packages** Additional technology-specific APIs, both standard APIs defined in Java Community Process (JCP) as well as non-standard device vendor or operator specific APIs.

Currently, most new mobile devices that claim Java ME compatibility, support *Connected Limited Device Configuration* (CLDC) and *Mobile Information Device Profile* (MIDP). CLDC provides low-level functionality and a virtual machine[8] whereas MIDP defines APIs related to display, local access and network among others.

There exist optional packages defined in JCP providing integration with web technologies. *JSR 226: Scalable 2D Vector Graphics API* [160] adds support for SVG Tiny 1.1 to Java ME to facilitate creation of rich user interfaces. Main use cases for SVG in Java ME are scalable images, animations, games and map visualizations [45, p. 10]. *JSR 287: Scalable 2D Vector Graphics*

[8]The K virtual machine (KVM) [139] is a small memory footprint version of the full Java Virtual Machine with reduced functionality, which runs in constrained devices.

*API 2.0* [161] is a draft version of the API, which extends the JSR 226 in a backwards compatible manner by adding support for SVG Tiny 1.2. Main additions over JSR 226 are the ability to create and modify animations. *JSR 290: Java Language & XML User Interface Markup Integration* [162] is another draft, which aim to enable creation of Java ME applications which combine web technologies used for creating UIs with Java code leveraging the W3C Compound Document Format (CDF) specification. However, the initial version is expected not to allow Java application integration in the browser environment due to technical issues related to scope and security model [162, section 5.2].

### .NET Framework

.NET Framework [42] is positioned to be the development platform for all new applications for Windows operating system, be it for the Web or for the desktop. .NET Compact Framework is a subset of .NET Framework available on mobile devices based on Windows Mobile operating system [39]. In addition, there exists a full implementation of the .NET Compact Framework for the Symbian OS [145]. The support for any .NET compliant language such as C# utilizing the same shared libraries is implemented using a virtual machine[9] which compiles the intermediate bytecode generated by compilers of each language into a machine code. The .NET Framework 3.0 consists of four major components[10], out of which Windows Presentation Foundation (WPF) forming the graphical subsystem of the .NET Framework is the most relevant in the context of this thesis [94, section 3.6.1]. Core functions of a web browser are supported by the .NET Compact Framework via Windows Forms WebBrowser control [43] which allows embedding web browser in .NET applications. Alternative mechanism to integrate .NET Framework to web browser is to utilize the web-based subset of WPF called Silverlight which is discussed in Section 4.2.4.

### Android

Google Android is a full mobile phone platform based on Linux kernel, which is expected to ship in devices in late 2008. In this section we discuss only the most relevant bits of its middleware layer related to the thesis.

[9]The virtual machine component of the .NET Framework in called *Common Language Runtime (CLR)*.

[10]Windows Presentation Foundation, Windows Communication Foundation, Windows Workflow Foundation and Windows CardSpace.

The Android includes a custom Dalvik virtual machine optimized for embedded OS which compiles Java code into a specific Dalvik bytecode. From this angle Android can be seen as a fork[11] of Java ME implementation with a better performance but with no compatibility with existing Java platforms or profiles. Related to platform integration with web technologies, there exists a mechanism[12] for binding a Java object to JavaScript running within the WebKit browser part of Android. It is also possible to control the DOM via Dalvik application using its WebKit integration. However, the initial version of the SDK does not describe the functionality related to WebKit integration to the JavaScript in detail, implying primary focus is on implementing applications in Java.

### 4.2.3 Web Browser as a Ubiquitous Client

A web browser is a software application for displaying and interacting with web pages build on web technologies. The core component of any web browser is its *rendering engine*, which tasks are to interpret the content and represent it to the user. The rendering engine[13] fetch and parse markup such as (X)HTML and construct it into the DOM. In addition, it constructs the boxes of content to render and applies style information, layouts the boxes and renders them. Finally, it interprets the system and user events and is responsible for the event-driven programming model by executing JavaScript code. The JavaScript program logic may modify the DOM and communicate over the HTTP with the server which is the core functionality of more interactive web applications to achieve rich interaction on the client-side. According to recent statistics [23], 97% of all the browsers utilize one of the four major rendering engines: Trident (Internet Explorer), Gecko (Firefox and Mozilla-based browsers), WebKit (Safari) or Presto (Opera). In Appendix B.1 an overview of the web standards support by rendering engines is represented. The *browser chrome* is the layer on top of the rendering engine which provides the browser UI components and additional functionality, which does not relate to web standards. Many browsers use the same rendering engine but differentiate by providing alternative browser chrome.

There exist various shortcomings in using the web browser as an ubiquitous client for applications compared to more established middleware platforms discussed in Section 4.2.2. First, there are major usability and interaction

[11]The core APIs and the virtual machine are not consistent with any Java platform.

[12]Using `addJavascriptInterface()` method of `WebView` class.

[13]Interchangeably, the term layout engine is used.

issues, namely poor I/O model provided by the DOM not especially applicable for desktop style applications in addition to page-centric update model with unsuitable semantics for applications such as *back*, *stop* and *forward*. Furthermore, security-related limitations restrict the implementation of browser-based applications to access multiple domains on the client-side. In addition, there is no access to local resources of the client from the web applications. Related to standards, the disregard of official standards and lack of standards in areas important to application such as more advanced networking, graphics or media capabilities has made the development of web applications tedious and error-prone. Closely related to this, performance issues have arisen as techniques and technologies are abused to implement functionality familiar from other platforms due to lack of better solutions. Despite all the above-mentioned shortcomings the web browser is increasingly used to implement applications. [143]

### Contemporary Web Browsers for Mobile Devices

The most advanced browsers for mobile devices are targeted to high-end device category which embodies modern operating systems (e.g. Symbian OS, Windows Mobile or Linux) and capable hardware. These browsers are increasingly utilizing the same rendering engines familiar from the major desktop browsers and incorporate advanced layout mechanisms suitable for small screens [172]. Currently, the major advanced browsers for mobile devices include Web Browser for S60 [44], Apple iPhone Safari [105] and Opera Mobile [176]. Both the Web Browser for S60 and the iPhone Safari are based on open source WebKit rendering engine whereas Opera uses its own proprietary solution.

### Web Browser for S60

Web Browser for S60 is the built-in web browser of the S60 platform. It is based on *WebCore* and *JavaScriptCore* components of the open source project WebKit [192], which is used in Apple's Safari browser[14]. Recently, WebKit has been also ported to other mobile platforms[15]. WebKit in turn, is based on the open source KHTML engine from KDE. The rationale for

[14]Precisely, WebKit is a system framework which is also used on Mac OS X by Safari, Dashboard, Mail and many other application.

[15]For example, WebKit is used in browsers found in Apple iPhone, Windows Mobile platform (`http://www.wake3.com/`), Java Micro Edition (`http://www.teashark.com/`) and Android.

selecting WebKit as the core for the browser for constrained devices stems from its standards compliance, performance, small memory footprint and license which allows it to be extended with proprietary components [159].

The Web Browser for S60 architecture is depicted in Figure 4.4 where the port of WebKit to S60 platform, S60WebKit [50] containing *WebCore* and *JavaScriptCore* from the WebKit, is emphasized. *WebCore* is the layout engine incorporating the HTML and CSS parsers, and HTML and XML DOM implementations taking care of the rendering logic. *JavaScriptCore* is the JavaScript engine responsible for interpreting and executing JavaScript source code. Rest the components within the S60WebKit are OS and S60 specific layers enabling the use of WebKit on the S60. Other component include the browser UI, the S60 Browser Control API for embedding the browser within native S60 applications, the plug-in API for extending the browser functionality and the HTTP framework for common network functionality [159, 172].

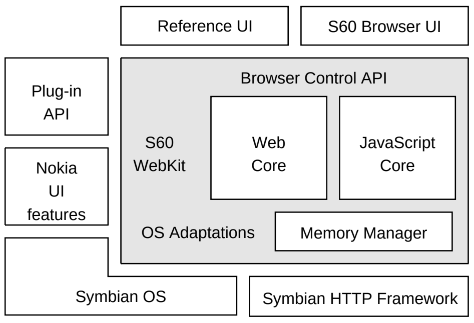


Figure 4.4: Web Browser for S60 architecture [50].

### Client-side Web Application Frameworks

Client-side web application frameworks discussed in this thesis refer to JavaScript libraries which execute within a standards compliant web browser. The framework components consist of one or multiple JavaScript files which are loaded similarly as any resources as a part of the web page. The frameworks can be split roughly to the following categories based on their features: *lower-level JavaScript frameworks* that provide a foundation on top of which new functionality can be build and *UI widget libraries* which provide richer user interface controls that abstract away native Web APIs and their incompatibilities across browsers. According to the survey done in 2008, Prototype[16] and jQuery[17] were the most popular lower-level JavaScript frameworks whereas Ext JS[18] and Script.aculo.us[19] were the most popular UI widget libraries [1]. Google Web Toolkit (GWT) [85] takes an alternative approach by allowing the development of web applications in Java which is translated to JavaScript before deployment to the Web by the GWT.

To summarize, although the frameworks differ in many ways, they all share some common architectural characteristics, namely they aim to: [136, p. 3]

- hide the complexity of developing interactive web applications which is a tedious, difficult, and error-prone task,
- hide the incompatibilities between different web browsers and platforms,
- hide the client-server communication complexities and
- achieve rich interactivity and portability and ease of development.

### 4.2.4 Web Runtime Environments

Rich client web applications or Rich Internet Applications (RIAs) discussed in Section 4.1.3 are executed within web runtime environments (WREs). This section discusses WREs which expand the functionality of the web browser. Such WREs can be seen as glue between the underlying middleware layer and the web technologies interpreted by the web browser.

The motivation for WREs is similar to that of shared libraries. By sharing such an environment across applications the amount of data that needs to be downloaded is reduced and less storage space is required on the target device. Furthermore, in the web context such environments typically expose additional functionality not provided by the standard web technologies such as richer graphics capabilities and mechanism for storing data persistently on the client.

From developers' perspective WREs enable the use of core web technologies to develop applications that interact with the underlying platform. This is typically implemented by providing JavaScript bindings to the underlying platform. Currently there exists no standard for such runtime environments

[16]`http://www.prototypejs.org`
[17]`http://www.jquery.org`
[18]`http://www.extjs.com`
[19]`http://script.aculo.us`

and on this account there are multiple incompatible WREs. Next, the most promising approaches are discussed.

### Microsoft Silverlight

Microsoft Silverlight is a web-based subset of Windows Presentation Foundation (WPF), the graphical subsystem of the .NET Framework discussed in Section 4.2.2. The Silverlight consists of two major parts. The *core presentation framework* handles the UI and user interaction and implements support for XAML, its DOM API integration and non-standard JavaScript APIs exposing Silverlight specific functionality. The *subset of the .NET Framework* contains components, libraries and a virtual machine for running code written in .NET compliant languages. The latter part providing .NET integration is scheduled to be incorporated in the upcoming version of the Silverlight. [138]

Silverlight applications use a web browser extended with a proprietary Silverlight plugin as the runtime environment. The core presentation framework supports rich animation, vector graphics and video capabilities. The Silverlight application UI and partly the interaction are implemented in XAML. In addition, JavaScript is supported and can be used to implement further interaction. The upcoming version of Silverlight is expected to incorporate a subset of the .NET Framework functionality which will bring the .NET language integration (e.g. C#) to Silverlight applications. In other words the developers of Silverlight applications can leverage the APIs familiar from standalone .NET desktop applications and interact with the DOM using such languages. The objects created in .NET compliant languages are also exposed to JavaScript. It has been announced that the Silverlight will be available to S60 during 2008 [53]. [194]

### Mozilla XULRunner

XULRunner is a cross-platform runtime environment for XUL applications, available for Windows, Mac OS X and Linux. It hooks up to the Mozilla Toolkit, which is a set of APIs built on top of Gecko and supports a wide array of web technologies, including XUL, XBL, (X)HTML, CSS, JavaScript, DOM, SVG and XSLT. Typically XULRunner applications utilize technologies such as XUL and XBL but they may use any language supported by XULRunner. The component-based engine architecture of XULRunner allows writing cross-platform, modular software which may leverage the shared libraries which reduces application size considerably. The bindings to shared libraries providing functionality such as file I/O and networking support is exposed to languages such as JavaScript using a cross-platform component model from Mozilla called XPCOM. Examples of popular applications built on XULRunner include Firefox web browser and Thunderbird email client. [76]

### Sun JavaFX

JavaFX refers to a suite of technologies for web application development by Sun Microsystems. JavaFX builds around existing Java technologies such as Java programming language and its Swing UI library. In addition, it includes a declarative scripting language, JavaFX Script, which is aimed to facilitate defining user interfaces for web applications. Its characteristics resemble more Java than JavaScript. Although being a dynamic language, JavaFX Script incorporates features such as static typing, support for classes, packages and inheritance. In addition, JavaFX Script can leverage the Java API which is part of the JavaFX platform.

The platform for running JavaFX application depends on Java runtime environment (JRE) similarly to Java ME. There exists a mobile implementation of the JavaFX called JavaFX Mobile which comprises of a full operating system for mobile devices capable for running JavaFX applications in addition to the JRE. [143]

### Adobe Integrated Runtime

Adobe Integrated Runtime (AIR) is a cross-platform runtime required for executing AIR applications which are build on top of Adobe's proprietary web technologies: Flash, ActionScript[20] and the Flex Builder development environment. The developers may use standard web technologies, (X)HTML, CSS and JavaScript to write applications. However, AIR runtime embodying the proprietary components is needed to execute such AIR applications. The runtime contains open source WebKit rendering engine for HTML layouting and Tamarin virtual machine for executing ActionScript and JavaScript. Due to this, AIR is independent of the web browser as it incorporates corresponding functionality within the runtime itself. Adobe's intent is to extend AIR from its current target platform, the desktop, to mobile devices in the future [29, p. 45]. [29]

[20]The Scripting language of the Adobe Flash authoring tool is a variant of JavaScript.

### 4.2.5 S60 Web Runtime – A Web Runtime Environment for the S60

S60 Web Runtime is the platform used for the concept implementation discussed in Chapter 5. On this account, a more thorough review of its features is given in this section.

#### Overview

The S60 Web Runtime (WRT) is a web runtime environment part of the S60 platform[21]. It supports building web applications called widgets using core web technologies, (X)HTML, CSS and JavaScript. The platform integration enables developing applications which leverage the S60 platform functionality using web technologies instead of native language of the S60, the C++.

WRT shares common components with the Web Browser for S60 with two notable changes. An additional layer implementing access to selected S60 platform functionality via JavaScript bindings is provided. Secondly, the browser UI, or the chrome, has been removed allowing developers to take total control of the UI. Consequently, there exist no mandatory UI elements such as navigation controls incompatible with application interaction models. From the architectural perspective, these changes are reflected to the topmost components in Figure 4.4: the *Reference UI* and the *S60 Browser UI* are replaced with a S60 Web Runtime specific component which wraps selected S60 platform functionality with JavaScript objects. To summarize, the S60 Web Runtime specific component provides the following features which distinguish applications, or widgets, leveraging it from web applications run within the Web Browser for S60: [51, 116]

- whole of the UI is implementable and customizable using web technologies
- access to platform specific functionality is provided via JavaScript bindings to selected platform APIs
- optional UI integration leverages native UI controls, such as built-in menu structures
- managed similarly as native applications, e.g. supports multitasking

[21]Starting from the S60 3rd Edition, Feature Pack 2.

- familiar deployment and installation model from native S60 applications

### Widget Components

WRT applications called widgets are distributed as ZIP compressed packages with an extension `.wgz`. These user-installable packages can be deployed to the target device via the web browser of the target device by downloading the `.wgz` package from the Web or by transferring the package to the target device from a desktop. The package comprises of the files presented in Table 4.1 out of which all the other file formats except the manifest are commonly used resources in the Web today.

Table 4.1: S60 Web Runtime widget components [52].

| File | Description |
|---|---|
| `info.plist` | Manifest file which contains metadata about the widget. (Mandatory) |
| `[name].html` | A standard HTML document containing the structure of a widget. (Mandatory) |
| `icon.png` | An icon of the widget visible in the application grid on the device. |
| `[name].css` | External style sheet files defining the presentational aspects. |
| `[name].js` | External JavaScript files implementing the logic. |
| `*.(jpg\|png\|gif\|bmp)` | Optional image resources. |

### Manifest

The manifest `info.plist` is an XML-formatted file which defines the widget metadata properties as key and value pairs and provides a declarative bootstrapping mechanism that enables widget engine to automatically instantiate a widget. An example of `info.plist` file properties is presented in Appendix A.2.

### Supported APIs

Since the WRT extends the Web Browser for S60 and utilizes its JavaScript interpreter and DOM implementation, all the APIs provided by the underlying environment are available to the WRT environment with some minor exceptions[22]. In addition, the standard APIs are extended with WRT specific JavaScript APIs. Overall, APIs exposed to WRT are:

**Core JavaScript APIs** The core JavaScript language. [75]

**DOM API** The API providing I/O model to the XML document which is fundamental to client-side JavaScript programming. [96]

**XMLHttpRequest API** The API for client-side functionality for transferring data between a client and a server. [118]

**S60 Web Runtime Specific APIs** The specific non-standard APIs exposing S60 platform-specific functionality to WRT. [52]

### S60 Web Runtime Specific APIs

The special features of WRT widgets are exposed to programmers via APIs which extend the standard DOM interfaces with widget specific functionality. These APIs are accessible via the following global objects: `widget`, `menu`, `menuItem` and `sysinfo`. `widget` object is a built-in module of the widget engine which provides methods for e.g. using persistent storage and controlling the navigation style. `menu` object provides an interface for manipulating the options menu and softkey bindings of a widget. `menuItem` object provides means to create menu items and child menu structures for options menu. System Information Service API (`sysinfo` object) is a plugin module which allows widgets to access certain subsets of system properties.

Some currently non-standard API extensions of the S60 Web Runtime [52] are being proposed for standardization. W3C Working Draft of *Widgets 1.0* [120] specification proposes some APIs of `widget` object to be standardized [120, Section 5.2], namely the methods for opening new browser windows and mechanism for storing and retrieving persistent data. The APIs to be standardized are similar to those of Apple Dashboard Widgets [107]. Next,

[22]Some properties specific to browsing context have been modified, for example due to lack of browser navigation controls in WRT, an alternative method for opening links is provided. [52, p. 11]

most important non-standard APIs supported by the WRT in addition to their standard-compliant alternatives – if available – are elaborated.

**Data persistence** For data persistence, the `widget` object provides methods `setPreferenceForKey()` and `preferenceForKey()` which allow storing and retrieving strings of data. The size of stored data is limited by the amount of free memory on the device [52]. On the Web, the most interoperable way to implement client-side persistence is to use HTTP Cookies [123]. However, the HTTP Cookie standard is severely limited. It allows only 4 kB of data to be stored into a one cookie and at most 20 cookies to be stored per domain [70, p. 460]. Google Gears [86] discussed in Section 3.6.3 provides less restricted persistence mechanism for major desktop browsers, but requires installation of a non-standard browser extension.

**UI integration** Mobile devices do not normally support desktop-centric concepts such as secondary mouse button. Furthermore, it cannot be expected that a cursor-based navigation method is supported at all with alternative input methods. For this reason most mobile device UIs utilize menu structures which are accessible via physical softkeys of the device. To be able to utilize this built-in functionality, WRT provides a `menu` object that can be manipulated via JavaScript to modify the menu structure. This functionality, however, needs to be emulated in environments with no native support for this feature by implementing an adaptation layer in JavaScript.

**Access to device capabilities** System Information Service API exposed via `sysinfo` object enables widgets to react to the device state changes, such as battery level of signal strength. The hooks are provided in a form of additional event handlers. In addition it allows controlling certain system functions such as alarm sounds, lights of the display or vibration. In order to use the API, a plugin module mush be loaded in the main HTML with the `embed` element[23]. The current WRT version 1.0 has limited access to advanced device capabilities through S60 platform such as GPS. However, it has been announced that the upcoming version of the WRT will expose more platform capabilities [51].

**XML data processing** In addition to WRT specific APIs, non-standard browser components `DOMParser` [70, p. 777] and `XMLSerializer` [70,

[23]The `embed` element was not included in any of the W3C (X)HTML specifications, however HTML 5 is legitimizing its use [98, section 3.14.4].

p. 940] originating from Mozilla ported to S60WebKit are exposed to WRT. The former is used to parse strings into DOM document objects and the latter to serialize them back to strings. Unfortunately, a Mozilla implementation of the XSLT Processor for manipulating XML files, `XSLTProcessor`, is not properly supported by WRT.

### 4.2.6 Other Platforms

In addition to platforms discussed above, there are dozens of other middleware platforms and web runtime environments. However, it is unlikely for most of them to be major commercial successes in the short term. For example, Maemo[24] and Qtopia[25] are Linux-based middleware platforms for mobile devices. OpenLaszlo[26] is a web runtime environment which has a unique capability to support different runtimes from the same codebase.

### 4.2.7 Overview of the Platforms

The operating systems were not particularly interesting in the scope of this thesis, as they all provided similar functionality expected of a modern operating system such as multitasking and networking support.

The middleware layer was found to be quite relevant in the context of emerging web applications which leverage the underlying platform capabilities. S60 platform supports web technologies via its web browser, although it was also possible to embed the browser component into a native application implemented in C++ as well [47]. Java ME was able to run across multiple operating systems and it supported subsets of selected web technologies through optional APIs. However, the application logic of Java ME applications similarly to Android was implemented in Java language which did not address the research question. Similarly, .NET Framework and its mobile specific derivative did require the base language to be one of the .NET compliant languages.

The use of a web browser as an application platform was discussed in Section 4.2.3. It was found out that client-side web application frameworks implemented in JavaScript are highly popular among web application developers. Such frameworks facilitate client-side application development within

[24]`http://maemo.org/`

[25]`http://trolltech.com/products/qtopia/`

[26]`http://www.openlaszlo.org/`

the web browser, but unfortunately they do not specifically take the constraints of mobile devices into account. On this account their use in mobile devices despite the availability of capable browsers is hindered by their less than optional usability and performance in mobile devices.

The major web runtime environments (WREs) discussed in Section 4.2.4 commonly incorporated the core components of the open source web browser rendering engine, the WebKit. However, only the S60 Web Runtime was targeting mobile devices whereas other WREs did not yet have versions for mobile devices publicly available. Furthermore, the other WREs relied heavily on proprietary languages for application development instead of utilizing standard web technologies.

Figure 4.5 depicts an overview of the reviewed client-side platforms and classifies them based on:

(i) the *runtime* environment executing the application logic and,

(ii) the *main programming language* used for constructing the client-side functionality of the application.

In the figure, the outermost circle, the *base platforms*, refers to the middleware layer used. Inner circle, the *web browsers*, illustrates platform leveraging a regular standards compliant web browsers. The innermost circle, the *web runtime environments*, depict environments which leverage the web browser or its rendering engine capabilities extended with the functionality of the underlying base platform. In general, the platforms on the inner circles leverage the capabilities of the platforms on the outer circle to some extend.

Table 4.2 compares the platforms discussed in this chapter based on their properties related to web technologies. *Platform features* columns indicates if the platform is targeted specifically for mobile devices, which *rendering engine* is used to interpret the UI and execute the logic and whether a mechanism for *interworking*, that is exposing platform capabilities to the rendering engine, is provided. The main languages used to implement *layout and styling* and *interaction* are defined under *main languages* columns.

## 4.3 Summary

This chapter discussed web applications and their characteristics and drew a taxonomy based on their interaction models. Secondly, the major software platforms on top of which such applications can be build were discussed and

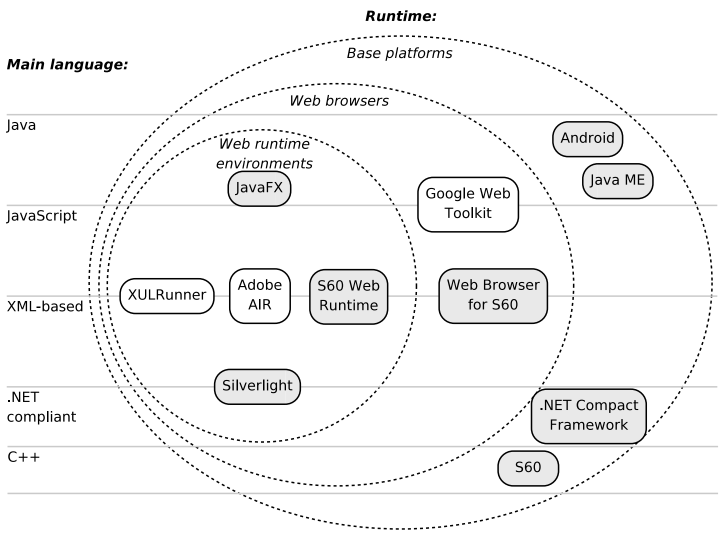


Figure 4.5: Classification of client-side platforms. Platforms specific to mobile devices are emphasized.

classified based on their features and web technology support. A specific focus was put on platforms specifically targeting mobile devices. Shortcomings were identified in using the web browser as an application platform. Regardless, it is increasingly the main target for application developers. To address the shortcomings, browser capabilities are being leveraged from other platforms. Namely, web technologies are being exposed to middleware platforms by means of embedding browser controls to native applications. On the other hand, emerging web runtime environments are embodying rendering engines as their core components commonly responsible for both UI and application logic.

To summarize, the integration of web technologies with more established platforms is still in flux, and the implementations are currently lacking many features commonplace in lower lever platforms and operating systems, such as file system access. This mainly stems from the lack of a standard security model for exposing platform capabilities to the sandboxed web context in a secure and interoperable way.

Table 4.2: Comparison of client-side platforms by their web technology interworking capabilities.

| | Platform features | | | Main languages | |
|---|---|---|---|---|---|
| Platform | Mobile platform | Rendering engine | Interworking | Layout and styling | Interaction |
| S60 | yes | WebKit | yes | C++ | C++ |
| Java ME | yes | - | planned | Java | Java |
| .NET Compact Framework | yes | proprietary | yes | .NET | .NET |
| Android | yes | WebKit | yes | Java | Java |
| Web Browser for S60 | yes | WebKit | no | (X)HTML, CSS | JavaScript |
| Google Web Toolkit | no | any | no | Java | Java |
| Silverlight | yes | browser plugin | yes | XAML | JavaScript, .NET |
| XULRunner | no | Gecko | yes | XUL, XBL | JavaScript |
| JavaFX | yes | browser plugin | yes | JavaFX Script | Java |
| Adobe AIR | no | WebKit | no | (X)HTML, CSS | JavaScript, ActionScript |
| S60 Web Runtime | yes | WebKit | yes | (X)HTML, CSS | JavaScript |

# Chapter 5

# Concept Implementation

From this chapter onwards, the thesis discusses the feed reader application called Feed Widget realized as the experimental part of the thesis. The following sections discuss the concept implementation starting with the use case description and functional requirements followed by a description of its architecture and components.

## 5.1 The Feed Widget – An Overview

The platform for the Feed Widget, the S60 Web Runtime (WRT) was discussed in Section 4.2.5 in more detail to provide the foundation for understanding its features and limitations. First, this section gives an overview of a concept implementation of the feed reader application which is entirely implemented in core web technologies, XHTML, CSS and JavaScript called Feed Widget. The main motivation for selecting the WRT as the platform for implementation was its full support for core web technologies for application development. The Feed Widget aims to deliver comparable level of functionality as the Web Feeds feed reader application and enhance it with features found in Google Reader, both of which were introduced in Section 4.1.4.

### 5.1.1 Functionality

The Feed Widget is an application that enables combining information from multiple web feeds together and provides an efficient and functional UI for displaying such information to the user. The semantical structure of the feed formats makes it possible to combine feeds from different sources together seamlessly. The Feed Widget support all major feed formats in addition to the Outline Processor Markup Language (OPML) used for interchanging feed aggregates discussed in Section 3.6.4. The main aim of the application is to provide a locally run application which creates a mashup of user-selectable web feeds. The main aggregate resides on the server which allows multiple client application to share the same data. Feed Widget leverages the native functionality of the WRT platform to provide better user experience than similar application running within standard web browser such as the Google Reader introduced in Section 4.1.4. This includes integration with the native controls such as built-in menus and storage mechanism for offline support. Regardless, the Feed Widget degrades gracefully to any standards compliant web browser[1]. The main functionality of the Feed Widget can be characterized as follows:

- Supports online feed aggregates via a RESTful interface
- Supports all major feed formats including various RSS versions (0.9x, 1.x, 2.0) and Atom
- Supports offline use by storing data persistently on the client-side
- Enables efficient filtering and sorting of information
- UI combines rich graphical capabilities which degrade gracefully to less capable user agents
- Integrates to the device-native controls and the web browser
- Support all major standards compliant web browsers in addition to S60 Web Runtime

### 5.1.2 Use Case Description

To set the Feed Widget in its context, the main use case is described below. The user stores his own feed aggregate to a web server, or uses an online reader application such as Google Reader [87], which publishes an aggregate of the feeds via a RESTful[2] interface. The user subscribes to the aggregate by entering the URI of the aggregate resource into the Feed Widget which

[1]Feed Widget has been tested with Firefox 2.0 and 3.0b2, and Safari 3.0.

[2]A RESTful system adheres to REST principles described in Section 3.1.

stores it persistently. Alternatively, the user may subscribe to any publicly available aggregate. This allows the user to use arbitrary software for managing his feed aggregate, provided that it provides means to publishing it in supported OPML aggregate format and that such resource is accessible over HTTP. The aggregate consists of feeds which in turn consist of messages which are displayed to the user as a mashup view which combines messages from different sources together. The mashup view displays headlines, publishing times and titles of the messages to the user. The user may sort the loaded messages by their publishing time or filter them based on whether he has already read them. Finally, the user may expand the mashup view to read a preview of the message, and open the full article in a new browser window at will.

### 5.1.3 Functional Requirements

Below the functional requirements for the Feed Widget are stated explicitly for the purpose of evaluation of the functionality of the concept implementation in the next chapter.

1. Must implement all of its logic on the client-side.
2. The server-side may be used for supplying static data in an XML-format.
3. Must function with a standards compliant web browser as a user agent[3].
4. Must provide a mechanism for storing resources persistently.
5. May use non-standard extensions for enhanced functionality if a fallback mechanism is provided.
6. Network interface must support HTTP and allow connections to arbitrary hosts.
7. Must support major XML-based feed (RSS, Atom) and aggregate (OPML) data formats.
8. The UI must adapt to various screen resolutions and aspect ratios.
9. The UI must be based on standard XHTML 1.0 elements, no custom tags permitted.

[3]Standards compliance is defined as a support for core web technologies, that is DOM Level 2, XHTML 1.0, CSS Level 2 and JavaScript 1.6.

### 5.1.4 Design Overview

The Figure 5.1 depicts the architecture of the Feed Widget where logical components are grouped into three tiers as introduced in Section 4.1.1 with a notable exception that all the components reside on the client-side. The presentation tier provides the interface to the user, the logic tier deals with data objects and modifies them and the data tier consists of a persistence mechanism which allows loading and storing data. The logic tier functionality can be further divided into two categories. The service layer provides services to the presentation tier, controls the application flow and modifies the data objects whereas the data access layer deals with the communication to the data tier.

The interaction model of the Feed Widget resembles that of rich client web applications depicted in Figure 4.1. The logic of the Feed Widget is completely implemented on the client-side and the server-side is only exposed as a open service API which provides data in a standard XML-based format to the client, depicted as a cloud in the Figure 5.1. Furthermore, a client-side persistence mechanism is used. The figure denotes that the Feed Widget is server agnostic; it communicates with any server in the Internet which can provide feeds and aggregates via HTTP using one of the supported Internet media types[4]. There is no stateful connection between the client and the server which makes the implementation fully REST compliant and decoupled from the server implementation.

#### Presentation Tier

The presentation tier is responsible for displaying the user interface to the user. In practice, the rendering engine of the WRT is responsible for rendering the UI according to the changes in the DOM. During the bootstrapping, the XHTML file defines the main container elements for views which provide hooks to the JavaScript implementation. The CSS which defines the presentational aspects of the Feed Widget is referenced from the XHTML and loaded during the bootstrapping process. All the presentational aspects of the application are defined within the static style sheets, which makes defining additional themes for the application straightforward. In addition, the WRT provides its own API to leverage native graphics capabilities of the device for richer graphical effects such as fade transitions which would be computationally heavy tasks if implemented by scripting the DOM. To fulfill

[4]Supported media types include `text/xml`, `application/xml`, `application/rss+xml`, `application/atom+xml` and `text/x-opml`.

the requirement for standards compliance, object detection is used to degrade gracefully to standards-based alternative in user agents not supporting WRT UI effects API.

#### Logic Tier

The logic tier is entirely implemented in JavaScript and is responsible for business and UI logic. The UI logic examines and modifies the DOM according to user and system events which is reflected automatically to the rendering engine. The rendering engine constructs the view according to the aforementioned changes in the DOM. In principle, the DOM interface serves as a large shared data structure between the logic tier and the presentation tier allowing flexible inter-tier communication. Network access is handled via a `Network` object which leverages `XMLHttpRequest` object. The WRT exposes non-standard functionality to JavaScript, namely mechanisms for parsing XML into a DOM `Document` object and serialized it back into text. Furthermore, the WRT relaxes the same origin policy allowing locally run web content to break out of the security sandbox to communicate with multiple hosts. Finally, logic tier abstracts and encapsulates the access to data tier into data access objects which enable using various storage methods via a single `Storage` interface.

#### Data Tier

The data tier stores the data used in the application in persistent stores. Feed Widget implements support for WRT persistent store [52, p. 12] in addition to HTTP Cookies [123]. The use of data access objects facilitates adding additional data sources. For example, Google Gears [86] relational database exposed to the browser context could be integrated to the system easily depicted with dotted lines.

### 5.1.5 Custom Components and Interfaces

The core components common to all WRT widgets were presented in Table 4.1 and APIs exposed to WRT were discussed in Section 4.2.5. In addition, some additional high-level functionality was implemented to facilitate the development. A summary of these additional classes[5] encapsulating commonly

[5]Although JavaScript does not provide a keyword for defining classes, the prototype-based inheritance can simulate such functionality to a sufficient degree.

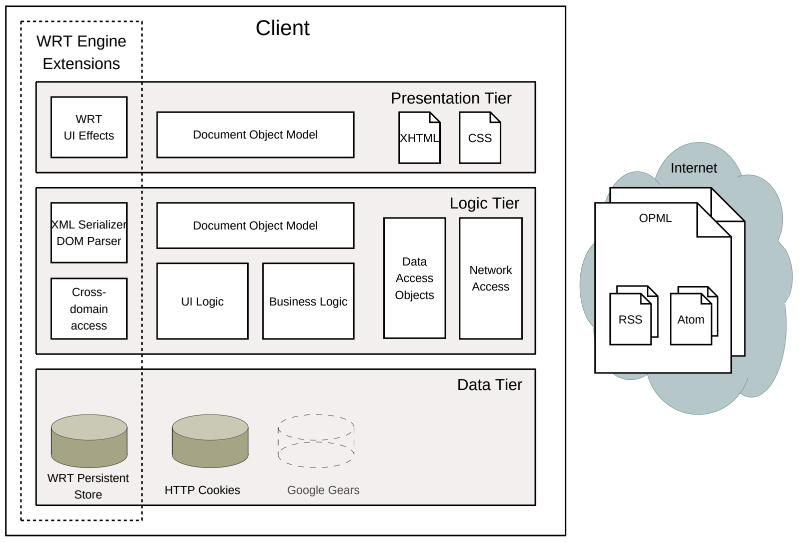


Figure 5.1: Architecture of the Feed Widget.

used functionality into reusable component in addition to extensions to the existing DOM interfaces is represented in Table 5.1. The main reason for implementing such component within the scope of this thesis and not leveraging existing web application frameworks discussed in Section 4.2.3 was that they did not support WRT specific functionality. Secondly, they were specifically targeted to desktops which introduced further inconveniences especially related to unsupported input methods and performance issues. Next, a quick overview of each custom component and DOM interface extension is given.

Table 5.1: Feed Widget JavaScript classes defining generic custom interfaces.

| File | Description |
|---|---|
| `global.js` | All utility function in the global namespace. |
| `EventDispatcher.js` | A class for dispatching events. |
| `Network.js` | A class providing Network-related functionality. |
| `Storage.js` | A class implementing persistent data storage. |
| `HTMLElement.js` | Extensions to the HTMLElement interface. |
| `String.js` | Extensions to the String object. |

### Global

Global component consists of functions defined within the global namespace. Due to the lack of a proper namespace mechanism in JavaScript, the use of the global namespace was kept in minimum to prevent namespace collisions which might occur if the Feed Widget would be extended with additional libraries. The main functions defined within the global namespace relate to DOM traversal: `$()` provides a convenient way to reference elements by their `ids` and `getElementByClass()` enables similar functionality based on elements' `className` property. Querying nodes based on their `className` properties was especially beneficial for interacting with a group of elements, for example to change their presentation. The above-mentioned two functions are highly popular among JavaScript developers and are contained within most of the JavaScript frameworks in some form. Regardless, there exist various different implementations. Fortunately, this may be remedied in the future as similar native implementations are being added in the HTML 5 [98] providing noticeable performance gains [168]. Despite the lack of native implementation, the utility of such functions override the associated performance penalties especially as the native DOM methods providing similar functionality do not provide any better performance.

### EventDispatcher

The `EventDispatcher` class provides a means of dispatching events. It receives events and calls appropriate actions based on the properties of the received events. Within the context of the concept implementation an action is defined as a method on the controller. The `EventDispatcher` supports using standard HTML element attributes `id` and `class` to declare actions the element should trigger. For example, once a valid event occurs on `<div id="doSomethingAction"/>` element it is automatically dispatched to the `doSomething()` method on the controller.

### Network

The `Network` class is responsible for all the HTTP communications. It leverages `XMLHttpRequest` object and extends its native functionality with a simple mechanism for caching, request timeouts and callback and exception handling. In addition, it supports storing and retrieving both text and DOM `Document` objects via the same interface. Advantage in using DOM `Document` objects is that they can be manipulated using the standard DOM API.

### Storage

The `Storage` class implements the Data Access Object (DAO) which abstracts and encapsulates the access to the data sources provided by data tier and provides a single interface for storing and retrieving data. Rationale for implementing such functionality was to provide a unified interface, a facade, to different data source APIs. This was especially significant due to unavailability of WRT specific data storage mechanisms on the development environment comprising of a Firefox web browser. The interface draws inspiration from the `Storage` interface of the HTML 5 represented in Listing 5.1.

Listing **5.1** Storage interface description [98, section 4.10.2].

```
interface Storage {
    readonly attribute unsigned long length;
    DOMString key(in unsigned long index);
    DOMString getItem(in DOMString key);
    void setItem(in DOMString key, in DOMString data);
    void removeItem(in DOMString key);
};
```

### HTMLElement

`HTMLElement` is the base interface implemented by all the DOM elements representing HTML elements. To simulate inheritance, the `prototype` property was used as a mechanism to extend the `HTMLElement` interface. This allows all the HTML elements to inherit the methods applied to it. The approach was enforced by its simplicity and efficiency, stemming from the fact that the `prototype` object is only referenced by the elements implementing the `HTMLElement` interface. Selected approach simulated inheritance sufficiently and allowed convenient access to utility methods extending the standard DOM. Only downside was incompatibility of the aforementioned extension mechanism with the WRT. Luckily this was remediable with a bit of JavaScript.

The rationale for implementing interface extension is obvious. When constructing a DOM tree using JavaScript, querying and modifying element properties, such as modifying the `className` property of the element to change its presentation is one of the most common tasks. Such actions may be triggered by event handlers fired by timed events, system or user actions. The approach, however, requires considerable amount of procedural code and is an error prone task due to the fact that the styles may be declared within the DOM via `style` attribute or by using external style sheets. As a solution to this and other similar repetitive yet error-prone tasks, convenience methods extending the `HTMLElement` interface were implemented.

Established JavaScript frameworks providing similar functionality were found to be desktop focused and as such were not applicable to the concept implementation. Due to this custom convenience methods extending the `HTMLElement` interface were implemented. These methods can be accessed via the `$(elem)` convenience function, for example, `$('foobar').show()` sets the element with an `id` of `foobar` visible. The implementation is inspired by various client-side web application frameworks, similar interface extensions have also been defined in HTML 5 [99, section 4.2] as well as in XBL 2.0 [199, section 7.2]. An overview of the extensions to the `HTMLElement` interface is shown in Table 5.2.

Table 5.2: Methods extending the HTMLElement interface.

| Method | Description |
|---|---|
| `add(type, handle, instance)` | Adds an event `handler` to respond to event of `type`, optionally pass the `instance` within which to execute the event handler. |
| `remove(type, handle)` | Removes the event handler. |
| `show()` | Sets the element as visible. |
| `hide()` | Hides the element. |
| `toggle()` | Toggles element visibility by using inline `style` property. |
| `setStyle({property: value})` | Sets the inline style of the element. |
| `getStyle(property)` | Get style of an element, both computer styles and inline styles are inspected. |
| `hasClass(className)` | Returns boolean indicating if the element has the class queried. |
| `addClass(className)` | Adds a class to the element. |
| `removeClass(className)` | Removes a class from the element. |
| `toggleClass()` | Toggles element visibility by using element's `className` property. |

#### String

The native JavaScript `String` object used to manipulate text was extended with methods which handle the ISO 8601 and RFC 2822 date formats commonly used in the feeds. It allows the dates to be converted to a consistent human-readable date format as well as to Unix time format suitable to be used as a key when sorting data structures. In addition, a method was implemented for stripping tags from the (X)HTML content which is commonly attached the feeds, but may cause unforeseen excessive data transfers due to referenced objects such as images.

### 5.1.6 MVC Implementation

#### Comparison to Standard MVC Model

Any MVC architecture enforces the principle of separation of concerns, that is a clear division between model objects that represent a model of the real world objects and the view objects that present the GUI elements seen by the user and the controller which ties everything together. However, implementing MVC application entirely on the client-side using core web technologies imposes some restrictions to the traditional MVC approach discussed in Section 4.1.1.

First, interaction in such an application is event-driven, that is the controller is responsible for both registering and acting upon DOM events. This is different from the traditional MVC design for client-server web applications which relies on URIs. Second, related to view, it is possible to directly interact with the DOM instead of using template-based approach for generating views typical in aforementioned MVC approaches. Such templating mechanism can be implemented on the client-side, but it requires leveraging technologies which have limited support on the client-side, such as XSLT.

Third, the models follow much the same pattern, they are just data structures mapped to some resource, typically loaded from the server. Models abstract the network interface and take care of asynchronous HTTP communication. Related to the concept implementation at hand, the fact that the client does not commit the modifications it makes to the model to the server makes the design simpler. However, in case such functionality would have been implemented there exists two popular data interchange formats for the purpose, the XML and the text-based representation for native JavaScript data structures, the JSON. The main benefit of the latter over the former is its seamless integration with the JavaScript.

Overall, the approach taken implements all of the application logic and components of the MVC on the client-side and uses the server as a simple open service API discussed in Section 4.1.2. Next, the components of the client-side MVC implementation are discussed in more detail.

### Main Components

For the model part, the concept implementation comprises of simple UI components such as buttons which have limited number of state information in them (e.g. visibility and a single value). In addition, there exist two entities modeling the encapsulated data, the feeds and the messages. This implies that the payback of implementing a separate model for each UI component as JavaScript objects is relatively small for the concept implementation and adds unnecessary complexity over utilizing build in DOM properties such as `className` and `value`. As an added benefit in leveraging the DOM as a model, there is no need to notify views of UI components of changes in the models as changes are automatically acted upon by the rendering engine. Furthermore, this approach makes it easier to persist the application as it is possible to simply serialize the DOM into a data structure if needed.

For the above-mentioned reasons, the DOM was considered to be a sufficient data structure for acting as a model for the UI components. On the other hand, for two main types of real-world object within Feed Widget, the feeds and the messages, separate JavaScript objects representing models were created as they did not map to any existing DOM elements with ease. Both the models will need their own views. The views of UI components evolved around standard XHTML elements such as `<button>` and `<div>`. A single controller was found to be sufficient for handling both user and system events and managing the overall program flow. The main functionality and related components of the MVC approach used were the following:

**Models** Models encapsulate business logic and domain objects. Models of feeds and messages are implemented as JavaScript objects which also handle interfacing with the `Network` and `Storage` objects.

**Views** Views comprise of the DOM, whose style is defined in CSS. Temporal dimension is implemented by leveraging CSS pseudo-classes and scripting the `className` and `style` properties of the DOM elements combined with timing functionality of the `window` object.

**Controller** Controller manages the interaction with the user interface by handling DOM event registration and dispatching events leveraging `EventDispatcher` object. Furthermore, it controls the program flow by intermediating between the models and the views.

Table 5.3 describes the components of the MVC implementation and the Figure 5.2 depicts the interaction between the components.

Table 5.3: Feed Widget JavaScript components implementing MVC.

| File | Description |
|---|---|
| `FeedWidget.js` | Feed Widget object instance. |
| `FeedWidgetController.js` | Feed Widget controller. |
| `FeedWidgetModelFeeds.js` | Feed Widget feeds model. |
| `FeedWidgetModelMessages.js` | Feed Widget messages model. |
| `FeedWidgetView.js` | Feed Widget view superclass. |
| `FeedWidgetViewFeeds.js` | Feed Widget feeds view subclass. |
| `FeedWidgetViewMessages.js` | Feed Widget messages subclass. |

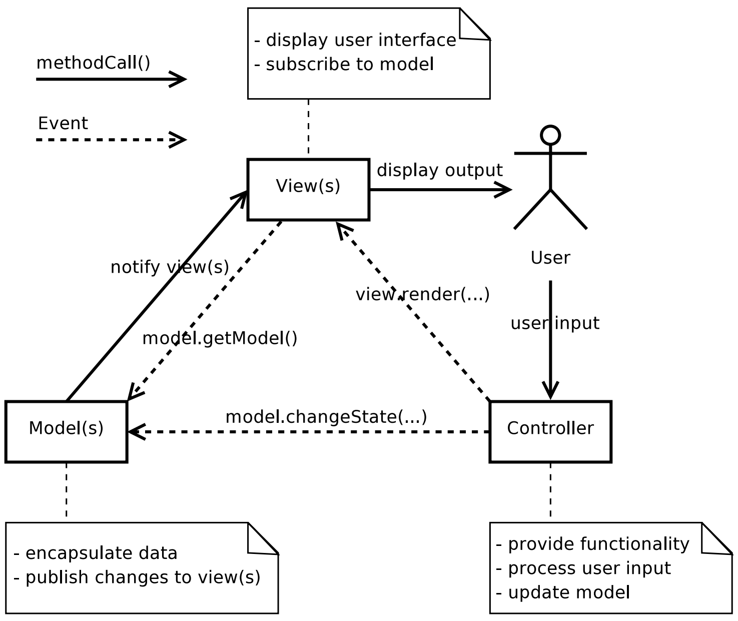


Figure 5.2: A diagram of the Feed Widget MVC structure.

### Bootstrapping and Layout

When the Feed Widget is started, the `info.plist` manifest file is inspected by the S60 platform to find out the capabilities granted for the application and to locate the main `[name].html` component to load initially. The structure of this XHTML file evolves around container `<div>` elements which enable parts of the UI to be moved using CSS without changing the structure of the page. A snippet of XHTML structure accomplishing that is shown in Listing 5.2. The `class` attributes denote initial states of the elements, e.g. `messagescontainer` element having classes `slidehide` and `right` is initially outside the visible viewport on the right and will switch roles with `feedscontainer` by sliding off the screen. By utilizing the standard `class` attributes a level of flexibility is archived using declarative markup to customize behavior implemented in JavaScript.

Listing **5.2** An example of the XHTML file bootstrapping the Feed Widget.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Feed Widget</title>
    <link rel="stylesheet" type="text/css" media="all" href="feedwidget.css" />
    <script type="text/javascript" src="js/util/global.js"></script>
    <!-- ... -->
</head>
<body id="body">
    <!-- ... -->
    <div id="topcontainer">
        <div id="messagescontainer" class="slidehide right"></div>
        <div id="feedscontainer" class="slideshow"></div>
    </div>
</body>
</html>
```

During the bootstrapping process the rendering engine constructs the initial DOM from the XHTML markup and loads additional referenced resources such as stylesheets, images and JavaScript files. Once all the referenced resources are loaded `load` event attached to the `window` object fires. Consequently, the `initialize()` method of the `FeedWidget` object is executed which instantiates the controller, models and corresponding views. Next, the controller executes its `loadFeeds()` method which populates the feeds model and passes it to the corresponding view to be rendered to the UI.

### Feed Widget Models

Feed Widget implements two model classes, `FeedWidgetModelFeeds` and `FeedWidgetModelMessages`. Both the models are instantiated by the `initialize()` method during the bootstrapping as described above. Both model components hold model specific properties and get and set methods for setting and returning those values. The most important method implemented by all the models objects is the `getModel()`, which returns the actual model objects[6]. The models interface with the network by leveraging the `Network` class.

### Feed Widget Views

Feed Widget has two views corresponding to models, defined in classes `FeedWidgetViewFeeds` and `FeedWidgetViewMessages`. Their main functionality is to render the view according to the supplied model and output it into the container, that is construct corresponding DOM elements and append them to the container elements in the DOM. This functionality is encapsulated within the `render()` method, which takes two parameters, the container element and the model object.

### Feed Widget Controller

The last but not least of the components is implemented by the `FeedWidgetController` class. It is responsible for managing the application flow and handling events. Listing 5.3 shows the `loadFeeds()` method of the controller used to load a feed aggregate which is displayed to the user as a list of feed items.

The logic of the `loadFeeds()` is the following. First, a reference to active Feed Widget instance is created as `instance` in addition to two nested functions `success()` and `failure()` defining the actions to take depending the success of the load operation. The fact that JavaScript allows functions to be used as data enables passing the aforementioned functions as parameters to the `getModel()` method of the model. Once the `getModel()` method is called, the model takes care of communicating with the server over HTTP leveraging `Network` object and constructs the model object. The passed

[6]To be precise the returned data type is an array which in JavaScript is an object with a thin layer of extra functionality. [70, p. 113].

functions retain their references to properties of `loadFeeds()`[7]. In case the HTTP request was successful, the `success()` function is executed within the context of the model and the `render()` method of the view displays the feeds to the user by creating the DOM elements and appends them to the `feedscontainer` element given as a parameter. Another parameter `feeds` is created by the model and can be referenced thanks to the extended scope chain of the nested functions in JavaScript. In case of a failure, the loading indicator will be hidden.

Listing **5.3** A method of Feed Widget controller used for loading feeds.

```
/**
 * Loads feeds and displays them
 * @param {string} uri An URI of the OPML resource
 */
loadFeeds: function (uri) {
    var instance = this.instance;
    var success = function (feeds) {
        instance.view.feeds.render('feedscontainer', feeds);
        $('loader').hide();
        $('splashcontainer').hide(true);
    }
    var failure = function () {
        $('loader').hide();
    }

    $('loader').show();
    instance.model.feeds.getModel(uri, success, failure);
}
```

### 5.1.7 Functionality Related to Views

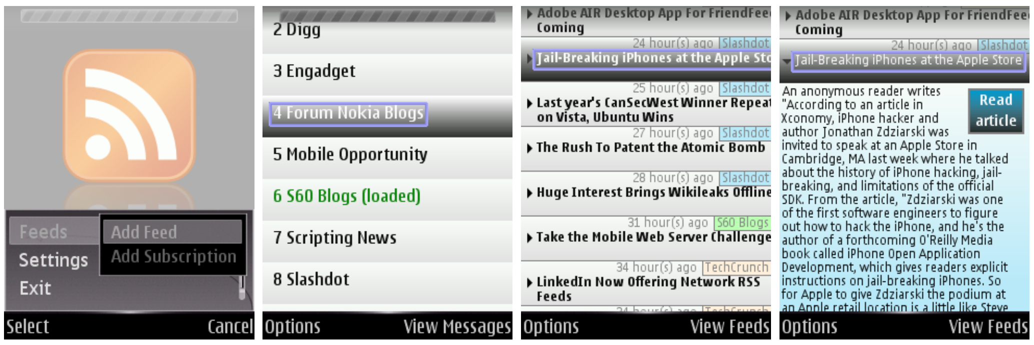
The view part of the application is the visible page which comprises of DOM elements that are rendered according to the XHTML markup and the style information within CSS. According to MVC, view has to provide an interface to the user from which to trigger events from and to update it once changes in the model occur.

As discussed above, during the bootstrapping of the application, the XHTML markup is loaded and a corresponding DOM is constructed. Next, style information contained within the style sheets referenced from the XHTML is applied to the corresponding DOM elements. The referenced style sheets are referred to as *computed styles*, that is styles that are applied before the element is displayed.

[7]This combination of code and scope is known as a closure. [70, p. 143]

After bootstrapping the application moves into the execution state, after which consequent edits to the views are only achieved by modifying the properties of the DOM via the interfaces it exposes to JavaScript. The DOM in turn is observed by the rendering engine which updates the rendering of the elements accordingly. This approach in which the DOM is exposed and manipulated through side effects is the only method in achieving higher interactivity in today's web applications. However, such a use of side effects has been discouraged for a long time in software engineering [142, p. 10]. In practice, the aforementioned approach was deemed to provide a sufficient performance for the basic UI manipulations such as adding and removing element or element properties. The main drawback was related to the performance overhead associated with recurrent DOM operations which prevented creating smooth animations by modifying the DOM.

In adherence to separation of concerns principle of the MVC, no inline event handlers or inline CSS style information were included in the XHTML. Interfaces defined in the DOM Level 2 Style module [195] were used to dynamically access and manipulate the style properties of the DOM and the DOM Level 2 Events module was used for registering the event handlers in the controller part of the application. Some examples of the Feed Widget user interface are presented in Figure 5.3.



Adobe AIR Desktop App For FriendFee Coming
24 hour(s) ago Slashdot
Jail-Breaking iPhones at the Apple Sto
25 hour(s) ago Slashdot
Last year's CanSecWest Winner Repeat on Vista, Ubuntu Wins
27 hour(s) ago Slashdot
The Rush To Patent the Atomic Bomb
28 hour(s) ago Slashdot
Huge Interest Brings Wikileaks Offline
31 hour(s) ago S60 Blogs
Take the Mobile Web Server Challenge
34 hour(s) ago TechCrunch
LinkedIn Now Offering Network RSS Feeds

Adobe AIR Desktop App For FriendFee Coming
24 hour(s) ago Slashdot
Jail-Breaking iPhones at the Apple Store
An anonymous reader writes "According to an article in Xconomy, iPhone hacker and author Jonathan Zdziarski was invited to speak at an Apple Store in Cambridge, MA last week where he talked about the history of iPhone hacking, jail-breaking, and limitations of the official SDK. From the article, "Zdziarski was one of the first software engineers to figure out how to hack the iPhone, and he's the author of a forthcoming O'Reilly Media book called iPhone Open Application Development, which gives readers explicit instructions on jail-breaking iPhones. So for Apple to give Zdziarski the podium at an Apple retail location is a little like Steve

(a) Menu (b) Feeds (c) Messages (d) Message details

Figure 5.3: Examples of the Feed Widget user interface.

## Locating Elements

In order to modify the view after bootstrapping, an efficient method for locating elements from the DOM was needed. To address this, each element was given a unique `id` property which was used as a fragment identifier and a reference to it was extracted via `$(id)` function. However, as the `id` property must be unique within a document [167, section 7.5.2], it was not appropriate for identifying multiple elements of a same type (e.g. element groups) – functionality that is needed to interact with repeating constructs. As a solution, the `className` property was used as a fragment identifier and queried with the custom global function `getElementsByClass()`. Re-using `className` properties this way was found to be more convenient mechanism to query elements compared to standard methods provided by DOM Core [125, section 1.2]. In addition, standard DOM API did not provide a simple way to add, remove and query `className` properties which had multiple classes. For this purpose, `HTMLElement` interface was extended with custom methods discussed in Section 5.1.5.

The main benefit in the design relying on `className` for querying elements was that it allowed elements to be positioned arbitrarily in the DOM as they were not defined in terms of their position in the tree. Furthermore, populating `className` attribute with multiple classes did not cause noticeable performance problems.

### Rearranging Elements

Rearranging the DOM by adding, removing and moving elements is a core functionality of the DOM API. The approach in the concept implementation was to declare the base layout providing appropriate `id` and `class` selector hooks in XHTML and further dynamically modify the DOM using its API. For example, Listing 5.4 shows how a new `div` element is appended to the element with an id `container` and subsequently removed from it.

Listing **5.4** Rearranging elements using standard DOM API.

```
// create a message element
var message = document.createElement('div');
// append the message element to the container element
$('container').appendChild(message);
// remove the message element from the container element
$('container').removeChild($('message'));
```

Alternative to standard DOM API is to use a non-standard `innerHTML` property [41] supported by major browsers. It is implemented by most of the HTML elements and can be used to append arbitrary (X)HTML markup to the DOM. This is typically faster and provides simpler solution in terms of lines of code compared to the standard DOM API. However, the flexibility of this approach is also its frailty as `innerHTML` allows adding malformed markup to the element which may cause unexpected results. For this reason, the concept implementation resorted in using the standard DOM API. Persistence across views was implemented by keeping the separate views in the same DOM while swapping their visibility or by moving them outside of the viewport using the absolute positioning of the CSS.

### Changing the Presentation

In the previous sections it was discussed how the `className` property of DOM elements was re-used for querying and describing their state. However, originally the property was specified to provide a mechanism adhering to separation of concerns principle where classes serve as hooks for separate presentational information declared within separate style sheets. This separation of the style and structure is a simple way to reduce the complexity of the application. Next methods used in changing the presentation of elements are discussed.

**Modifying style sheets** The cascading style sheets comprise of computed styles that are added to the element beforehand. Typically the style sheets are kept static, although DOM Level 2 also defines an API for querying, traversing and manipulating style sheets themselves. However, it is not widely supported by major browsers [70, p. 383].

**Switching the `className` property** Popular mechanism used to change the presentation of elements after the DOM has been constructed is to use script the `className` property. Some problems were faces with the standard DOM API in cases when the computed styles from the static style sheets were overridden by those of `style` properties which have the highest priority in the cascade. As a solution, utility functions `getStyle()` and `setStyle()` were implemented which resolve this conflict by prioritizing presentation set via `className` property over other means.

**Modifying the `style` property** To fulfill more advanced interaction requirements such as animations for sliding elements smoothly from side to side requiring unforeseen modifications, it is not feasible to use predefined static classes. The main method for implementing such functionality was to modify the `style` property of an element via JavaScript utilizing timers provided by the Browser Object Model for recurring updates, namely `setTimeout()` and `setInterval()` methods of the `window` object. Similar functionality was implemented declaratively leveraging experimental CSS Transitions module [103].

**CSS pseudo-elements and -classes** The pseudo-elements and pseudo-classes of CSS Level 2 discussed in Section 3.3.2 provided accessibility benefits, for example emphasizing actionable elements. However, they did not provide rich enough mechanisms for implementing dynamic behavior needed by e.g. master-detail functionality as the pseudo-classes allowed only modifying the CSS classes of the focused element. [18, section 5.11.3]

### 5.1.8 Event-driven Design

The controller acts as an intermediary between the model and the view which are decoupled from each other. The controller of the concept implementation resembles desktop GUI applications in a sense, that it takes care of registering and acting upon events attached to the elements in the view triggered by the user or system actions. As an ability to manipulate events is integral part of any interactive web application, the design approach taken with the event model is discussed next in more detail.

In general, there exist three different event models: simple DOM Level 0 API codified by the HTML 4 standard [167], the W3C DOM Level 2 event model [96] and the non-standard event-model of the Internet Explorer [70, p. 388]. The concept implementation utilizes the more powerful DOM Level 2 event model defined in [156] due to its following unique characteristics:

- supports event propagation, that is a mechanism which allows also ancestors of the event target to act upon events in addition to the source,
- allows registering multiple event handlers to a one element, in compliance with the MVC [70, p. 399] and,
- works on any DOM element compared to just HTML elements of the Level 0 API.

#### Event Registration and Propagation

All types of events triggered by user, system and network actions were centrally managed within the controller. The registration and removal of event listeners was handled by `addEventListener()` and `removeEventListener()` methods defined in `EventTarget` interface which is part of standard DOM API. In addition, standard event types defined in `HTMLEvent` module were utilized. Thanks to event propagation mechanism, there was no need to register new event handlers each time new elements were added to the DOM. For example, once the `keydown` event was fired on a key press, the ancestors of the event target had an ability to act upon the event prior to target itself[8].

The events are first acted upon by the `EventDispatcher` class whose main role is to dispatch events to actions. The actions are methods of the controller which define simple domain logic to interact with the models and the views, such as get the model and ask the view to render it accordingly. More complex domain logic is contained within models in accordance with the MVC best practices. Two significant design choices related to event handling mechanism implemented are discussed below:

**Binding events handlers declaratively** To simplify the design, standard attributes of the HTMLElement interface `id` and `className` were reused to declare which action should be executed[9]. This means, that actions may also be modified declaratively by defining attributes from within the XHTML. In case actions will be bound to repeating or nested construct which do not have unique actions defined, the `className` attributes are used instead.

**Listening events high up in the DOM** By attaching the event listener near the document root has multiple benefits. Instead of registering arbitrary amount of event listeners to each active element, one can simply attach a listener to one of the ancestor elements, even to the `document` root, and inspect the event object which includes further details of the event such as the reference to the element which triggered the event. The use of the event propagation mechanism greatly simplified the design of the implementation. The most noticeable benefit in the concept implementation was that there was no need to add event listeners to the newly created active elements. [156]

It was deemed that the above-mentioned features, the listening of event centrally and the declarative binding mechanism facilitated designing the appli-

[8]Precisely, the DOM Level 2 event propagation has two phases. In capturing phase the event propagates from the root of the document to the target, and in bubbling phase the event bubbles back up from the target to the root.

[9]To distinguish elements which trigger actions, the identifiers of such elements use lowerCamelCase naming convention.

cation towards the goal to centrally manage the entire event handling logic within the controller. An example of the XHTML code used to register an event handler declaratively and associated `MyController` controller object defining the event handling code in addition to simplified `EventDispatcher` class is represented in Appendix A.3.

## 5.1.9 Models Representing the Domain Objects

The model components represent the domain objects of feeds and messages and implement associated business logic. As the focus of the concept implementation was more on the interaction and user interface side, the models were kept relatively simple. Regardless, an overview of the main issues related to the models is given.

### The DOM as a Multipurpose Data Structure

The DOM properties held some element level properties of the model for UI elements, such as information of the element state declared within the `className` property of the element (e.g. `open` or `closed`). Similarly, actions the element should fire on events were defined within `id` or `className` properties of elements. This design approach was adequate for the concept implementation which was limited in complexity. However, when dealing with more complex UIs it may be desirable to store such information within its own model object which references the view instead of leveraging the DOM properties. Within the scope of this work, the performance penalty of extensive use of the DOM was not studied, however no slowness was perceived.

### Native Data Structures for Model

The application models take an URI of the XML resource as a parameter and return the resource converted to a native JavaScript object. To accomplish this, first the model delegates network communication to the `Network` object which fetches the XML resource and converts it into a DOM document object. Next, the model's `toModel()` method traverses the DOM document object using the standard DOM API and saves relevant tuples into an array which holds the model properties. Prior the insertion into the array each tuple is validated and unnecessary data is removed. For example, redundant (X)HTML markup contained within the message descriptions is converted to plain text.

The time it took to validate the data was considerably smaller than traversing the full DOM document object multiple times – a process which is done while reflecting the changes in the data model back to the DOM constituting the view part of the application. Moreover, traversing DOM is known to require considerable amount of computational time and memory[10] and it was deemed that DOM is only used as a data structure for view. Furthermore, inherent benefit in using an array over DOM as a data structure was that it could be searched in constant time.

## 5.2 Summary

This chapter focused on the concept implementation design and its architecture. In addition, selected issues were discussed in more detail. The concept implementation was completely implemented in client-side web technologies, mainly JavaScript, and adhered to established design practices in software development such as MVC and separation of concerns. The implementation presented a novel way of applying these practices into the client-side web application development and discussed main challenges discovered. In addition, a rudimentary JavaScript framework and supporting library was created to facilitate client-side web application development on the target platform.

[10]The DOM is 2 to 5 times the size of its representation in XML [146, p. 174].

# Chapter 6

# Evaluation

In this chapter, the Feed Widget concept implementation is evaluated against the key requirements defined in Section 2.5. The evaluation was conducted by comparing each of the interaction pattern and user interface language requirements defined in Sections 2.5.2 and 2.5.3 respectively with respect to the generic requirements defined in Section 2.5.1. First, the results of the evaluation are represented followed by reasoning each requirement to justify the evaluation results. Discussion and analysis of the generic observations categorized according to the generic requirements follows in Chapter 7.

## 6.1 Overview of the Results

Table 6.1 represents a summary of the evaluation incorporating all the key requirements defined in Section 2.5. The columns represent generic requirements $R_{11}$ through $R_{15}$. The interaction pattern and the user interface language requirements are depicted on the vertical axis and evaluated against the generic requirements listed on the horizontal axis.

The requirement $R_{11}$ *Functionality* is evaluated as 'no' if the given requirement was impossible to fulfill, 'yes' if the requirement was fulfilled and 'part.' if the requirement was fulfilled only partially. The other generic requirements $R_{12}$ through $R_{15}$ are not evaluated if the functionality was not implemented.

A subjective estimate represented as one to three plus signs '+' is given for each requirement $R_{21}$ through $R_{27}$ and $R_{31}$ through $R_{36}$ respectively. In all the cases, estimate '++' sits between the extremes. Empty refers to not applicable.

In $R_{12}$ *Efficiency*, '+' implies that the perceived performance is below and '+++' above that of a similar native S60 implementation.

In $R_{13}$ *Portability*, an estimate of '+' is given if the porting of the functionality to standards compliant browser requires device recognition using procedural code or proprietary extensions. An estimate of '+++' is given if portability is a built-in feature of a declarative language or if no adaptation is needed to fulfill the requirement.

In $R_{14}$ *Ease of authoring*, '+' means that mainly procedural code is used, while '+++' implies that the functionality is realizable with the built-in declarative constructs specified for the purpose.

In $R_{15}$ *Web integration*, if proprietary technologies are predominantly utilized, the estimate is '+'. If only open web standards and APIs are used, '+++' estimate is given.

The sum of the estimates roughly represents the capability of the WRT platform and web technologies it supports (see Section 4.2.5) for the implementation of the given requirements. Overall, the evaluation results estimate the applicability of the WRT for implementing applications which incorporate functionality common for native applications built in device-dependent languages using web technologies only. While interpreting the results, it should be noted that the evaluation has been conducted from the viewpoint of the concept implementation only and thus cannot be generalized.

## 6.2 Interaction Pattern Requirements

### 6.2.1 Typical Interactors

In order to expand the array of interactors available in (X)HTML, links and form controls, the concept implementation resorted in using JavaScript. By leveraging JavaScript virtually any (X)HTML element may be active in a way that it triggers further system actions. This was implemented by leveraging the DOM Level 2 Event model as discussed in Section 5.1.8 which allows expanding the array of interactive elements to cover labels, lists and icons among other standard elements. The CSS pseudo-classes such as `hover` provide an ability to style such active elements distinctively.

Related to the interaction with such elements, support for key events was implemented so that they could fire on any element. There existed three keyboard event types `keydown`, `keypress` and `keyup`. However, the `keydown`

Table 6.1: Evaluation of the concept implementation against the interaction pattern ($R_{2n}$) and user interface language requirements ($R_{3n}$) with respect to the generic requirements ($R_{1n}$).

| | | Generic Requirements | | | | |
|---|---|---|---|---|---|---|
| | Requirements | $R_{11}$: Functionality | $R_{12}$: Efficiency | $R_{13}$: Portability | $R_{14}$: Ease of authoring | $R_{15}$: Web integration |
| Interaction Pattern | $R_{21}$: Typical interactors | yes | ++ | ++ | ++ | +++ |
| | $R_{22}$: Master-detail | yes | ++ | ++ | + | +++ |
| | $R_{23}$: Paging and dialogs | yes | ++ | + | + | ++ |
| | $R_{24}$: Repeating and nested constructs | yes | + | ++ | + | +++ |
| | $R_{25}$: Copy-paste and undo-redo | no | | | | |
| | $R_{26}$: Drag and drop | no | | | | |
| | $R_{27}$: Filtering | yes | + | + | + | +++ |
| UI Language | $R_{31}$: Graphically rich content | part. | + | + | ++ | ++ |
| | $R_{32}$: Exact control of the presentation | yes | ++ | +++ | ++ | +++ |
| | $R_{33}$: Layout and content adaptation | part. | +++ | ++ | +++ | +++ |
| | $R_{34}$: Navigation | part. | ++ | + | + | ++ |
| | $R_{35}$: Interactive graphical menu system | yes | ++ | + | + | + |
| | $R_{36}$: Customizable presentation | yes | +++ | ++ | +++ | +++ |

was the only event type consistently implemented across browsers. By leveraging event propagation, the Feed Widget provided means to centrally act on events defined declaratively. Identifier of an event handler was attached to either `class` or `id` attribute of an element as a string of text. Similarly, semantic meaning was added to interactors using e.g. classes `open` and `closed` to represent its states.

Although key events are not standardized in DOM Level 2 Events module, they were supported sufficiently across browsers used for development. The approach allowing to catch all the events independent of the target element event registration state was deemed to be the most versatile approach as it kept the event handling code simple and centralized. The event propagation did not cause any problems with the performance.

Despite some issues, dealing with typical interactors was deemed relatively effortless. Furthermore, functionality of a declarative language was used to identify event handlers.

### 6.2.2 Master-detail

Master-detail type of interaction was used in expanding feed view to messages view and in revealing the extended view containing message description. There existed no declarative way for defining such interaction as the CSS pseudo-classes did not reflect their state to any of the DOM properties. On this account JavaScript was used to realize this functionality via scripting the DOM. Declarative approach was pursued to the greatest extend possible by only manipulating appropriate `class` attributes of the elements. Fortunately, the upcoming HTML 5 defines a `details` element providing a declarative way to implement similar functionality [98, section 3.18.1].

Overall, there were no issues other than high reliance on JavaScript for implementing the logic, which affected the ease of authoring.

### 6.2.3 Paging and Dialogs

For paging, the concept implementation used container elements which were swapped in and out of the visible viewport. As the web technologies have not been particularly designed for application UIs but for navigating across pages, there exists no standard way to implement application specific navigation mechanism comparable to the traditional web page centric model used in web browsing.

For dialogs, it was found out that legacy JavaScript `window` object methods `alert()` for displaying dialog box, `confirm()` for displaying a question and `prompt()` for asking user input integrated with the device native menu seamlessly. Using these methods in desktop browsers is commonly considered a bad practice as they generate disruptive popup dialog boxes. However, due to integration with the native menu of the mobile device, this was not deemed disruptive.

Due to the above-mentioned reasons and the lack of a proper and standard way to implement paging and dialogs, portability and ease of authoring was ranked lower.

### 6.2.4 Repeating and Nested Constructs

Web applications commonly contain repeating sections or blocks, for example for data entry of multiple items or a view of messages in which each message contains a title, description and date which is replicated multiple times such as in the Feed Widget. The core web technologies did not support defining repeating or nested constructs without resorting to scripting. HTML 5 is defining a repetition model which can be implemented with no scripting [98, section 7]. Due to lack of support for HTML 5, the Feed Widget relied in generating such constructs from data models and programmatically appending such elements to the DOM by means of scripting.

The efficiency of this functionality suffered from the extensive use of the DOM operations to manipulate document structure. Furthermore, implementing such functionality was deemed somewhat tedious, as no declarative mechanism was available.

### 6.2.5 Copy-paste and Undo-redo

Copy-paste and undo-redo are common interaction patterns from the desktop software. Yet there exists no standard mechanism for implementing them using web technologies. Actually, they are typically implemented by the user agent which leverages the built in functionality provided by the operating system. Such functionality was not exposed to the WRT and implementing support for these patterns would have required modifications to the WRT and the S60 which was deemed to be outside of the scope of this thesis.

### 6.2.6 Drag and Drop

Drag and drop was not implementable as there was limited support for mouse events in the WRT as discussed in Section 7.3. ARIA proposes properties for drag and drop that describe drag sources and drop targets which could be leveraged to implement such functionality. However, implementing this would require modifications to the WRT which does not incorporate support for ARIA.

### 6.2.7 Filtering

Filtering pattern typically refers to an action in which the user limits or sorts a dataset consisting of multiple items of same type. To implement such functionality with the core web technologies, JavaScript is mandatory. The Feed Widget implementation revealed that a common use case for filtering was not implementable in the WRT. Namely, the free-text search with autocompletion did not function root cause being the inability to fire `change` events on the input element text content changes. To implement sorting of DOM elements for the purpose of ordering e.g. sibling elements of a container element, the most straight-forward way was to implement a custom sort function which compares the properties of elements selected as sortable properties and appends the elements to their new location. This allows elements to be live in the DOM all the time and in theory they may act on events during the sort operation. However, it was observed that implementing such custom functionality may introduce performance problems if the sort algorithm is not optimized for the task.

The efficiency and ease of authoring findings related to the repeating and nested constructs defined in Section 6.2.4 apply similarly to this functionality. Furthermore, implementing sorting functionality in JavaScript affected the efficiency negatively. In addition, the inability to support auto-completion affected the portability of the solution.

## 6.3 User Interface Language Requirements

### 6.3.1 Graphically Rich Content

Using basic primitives such as lines, rectangles, images and text was well supported in CSS and it was deemed that leveraging CSS for basic UI authoring was effortless. On the other hand, the inherent characteristics of the CSS box model [18, chapter 8] did constrain the design to rectangular elements. Due to the fact that SVG was not supported by the WRT, there was no standards compliant way to leverage vector graphics. Raster graphics was used sparingly mainly for decorating the UI elements to minimize the application size.

For implementing graphically rich temporal interaction, three approaches were taken. First, the most standards compliant way was to leverage JavaScript timers to schedule code to be executed at certain point of time or time intervals. `setTimeout()` and `setInterval()` methods of the `window` object were used for this purpose. However, their use was kept to minimum as the extensive use introduced performance problems and led to unresponsiveness of the UI. Second, WRT-specific `widget` object methods `prepareForTransition()` and `performTransition()` were used to provide hardware accelerated fading effects for the UI elements. Third, an experimental CSS Transitions module discussed in Section 3.5.2 was used which enabled similar functionality without resorting to scripting. The CSS approach was considered the most applicable due to its backwards compatibility, performance and ease of authoring.

Efficiency and portability were causing most challenges. Efficiency was modest due to the use of JavaScript to implement interactive graphical effects. Portability was challenged due to use of proprietary functionality of the `widget` object.

### 6.3.2 Exact Control of the Presentation

The CSS was used for defining the presentation which required both the absolute positioning of selected elements and scalability from the layout. Related to layout, it was deemed that declaring vertical placement of elements was cumbersome. For example, placing element just above the visible viewport was unintuitive. Furthermore, as a single element cannot have multiple backgrounds[1], one has to resort to using wrapper elements which unnecessarily complicate the DOM degrading the performance.

The lack of a proper SVG implementation in the WRT made designing graphical UI elements which scale to any screen resolution and aspect ratio more tedious. Regardless, CSS provided sufficient support for defining scalable layouts but it lacked a mechanism for defining other than rectangular shapes which scale seamlessly stemming from the limitations of the CSS box model. For example, there existed no means to use different sizes of raster images depending on the device characteristics. The above-mentioned limitations of the graphics sub-system, mediocre performance and the lack of 2-dimensional vector graphics model not bound to the DOM model are being tackled by the `canvas` tag and associated API defined in HTML 5 [98]. This block level element providing a canvas to which graphics can be drawn represents an adaptation of a typical system level library part of operating systems and middleware to the web environment.

The lack of SVG support limited the functionality, and thus the functionality was fulfilled only partially. Nevertheless, portability and web integration were deemed to be the two most notable benefits of using web technologies for defining the presentation. First, the CSS did have built-in functionality to support either absolute or relative positioning despite some specific issues discussed above. Secondly, no procedural code was used contributing to the ease of authoring.

[1]As of CSS Level 2, this is addressed by CSS Level 3 Background and Border module.

### 6.3.3 Layout and Content Adaptation

The layout of the Feed Widget was designed to scale to any screen resolution and aspect ratio common to modern mobile devices and desktops. This was implementable using relative positioning and sizing and by using only CSS for layout and styling. However, related to content adaptation, there existed no standard mechanism for providing alternative or fallback content for e.g. images of different sizes. SVG which was not supported by the WRT would solve the scalability issue of vector graphics but for raster graphics, video and audio elements there existed no solution for adaptation on the client-side.

CSS Level 2 media types discussed in Section 3.3.2 provide a trivial mechanism for delivering alternative style sheets based on device type. For adapting to more specific characteristics such as screen size or aspect ratio, media queries of CSS Level 3 discussed in Section 3.5.2 provide a robust solution. Unfortunately, the former is defined unambiguously and the latter is not yet supported by the WRT or most of the major browsers. Some functionality of the media queries can be implemented using JavaScript such as adaptation to various screen sizes. However, such information is courtesy of the de facto `window` object and cannot be completely relied on due to varying implementations.

Despite highlighting the limitations above, the benefits of leveraging web technologies are obvious for adaptation. Especially designing adaptable layouts is straightforward using built-in constructs of the CSS. Furthermore, textual content such as text is easily adapted to different device characteristics. However, portability of interactive content is not supported sensibly.

### 6.3.4 Navigation

The WRT provides two mechanisms for navigation, that is focusing and activating elements which have events attached to them. First method called pointer navigation, uses a pointer (i.e. a cursor) which emulates a mouse. Another method is known as the focus navigation method and it focuses only on focusable elements with no visible pointer. The latter model is somewhat similar to the model proposed in WICD [134, section 6.2].

Some incompatibilities with the navigation mechanism of the WRT related to the DOM Level 2 event model were confronted. The main problems related to the focus navigation mode. In this mode, the event propagation functioned properly only if the event listeners were added to the `document` object representing the root node of the entire HTML document. Furthermore, the only event type properly supported by the focus navigation mode was `keydown`. For compatibility with the pointer navigation and desktop user agents with a mouse, another event listener was added accordingly which listened for `click` events. Other event types were found to be not supported consistently across the WRT and major browsers. The coexistence of the two parallel event listeners did not cause any problems.

Another limitation of the focus navigation was that only limited set of (X)HTML elements gained focus automatically, namely anchor (`a`) and `input` elements. As the event handling mechanism of the Feed Widget relied on event propagation rather than on attaching event listeners directly to elements, a solution was to attach dummy DOM Level 0 event handlers directly to those navigation elements not focusable by default. The above-mentioned limitations emphasize that there is a room for improvement regarding the event handling mechanism of the WRT, especially while using the focus navigation.

Navigation scored lower in portability and ease of authoring due to the fact, that it used proprietary focus navigation method, in addition to the above-mentioned limitations.

### 6.3.5 Interactive Graphical Menu System

The concept implementation utilized the `menu` and `menuItem` objects of the WRT which provide means to utilize the built-in native menu structures of the S60. However, to be interoperable, such functionality must be emulated in other platforms. Wireless Markup Language (WML) introduced a somewhat similar mechanism for leveraging device resident keys for common navigation tasks. However, it has been superseded with XHTML Mobile Profile for constrained devices and with (X)HTML for more capable devices which defines an `accesskey` attribute that can be used somewhat similarly. However, currently non-standardized built-in OS and windowing system key shortcuts override accesskey settings which undermines its usefulness as a cross-platform solution. For similar reason, the WRT does not support `accesskey` attribute which has been deprecated it in XHTML 2 in favor of the Role Access Module [5, section 25]. Currently there exists no markup attribute for defining key shortcuts in an interoperable manner. This is deemed a drawback as it forces developers to implement their own custom key event handling mechanisms at the cost of accessibility.

Out of all the requirements, interactive graphical menu system scored lowest although the proprietary objects which provide access to built-in native menu provided the same level of efficiency as the native implementation. However, the implementation was not portable, and did not integrate well with the existing web technologies due to its proprietary approach. Furthermore, it was deemed that the API used to interact with the menu system was clumsy.

### 6.3.6 Customizable Presentation

Only externally referenced CSS resources were used for defining the presentation of the Feed Widget. This allowed the presentation to be customized simply by adding another CSS file which requires no changes to the program logic or any other components. To reflect the style sheet changes during the execution, `disabled` property of the `link` element which is used to load the external CSS files provided a mechanism for switching style sheets fulfilled the main use case for customizable presentation. It allows, for example, to switch to alternative presentation or theme during the execution of the Feed Widget. An alternative method for accomplishing similar functionality assessed was to leverage DOM Level 2 Style API which allows manipulating style sheets programmatically during the execution [195]. However, this API was not yet implemented across browsers.

The customization of the presentation was well supported by the CSS. Virtually any element could be styled within the expressiveness of the language. Only the above-mentioned limitations in the support of some advanced functionality related to programmatic manipulation affected the portability.

## 6.4 Summary

In general, most of the *functionality* was successfully implemented. The *efficiency* of the implementation was deemed below that of native implementation mainly in cases where extensive use of JavaScript was needed to implement rich graphical effects such as animations, otherwise the performance was not an issue. *Portability* was sufficiently built-in to standard web technologies excluding limited support for portable graphically rich content, navigation and interactive graphical menu system. *Ease of authoring* was ranked lower due to the fact that JavaScript was used to implement most of the functionality of the concept implementation due to lack of expressiveness in declarative languages used. Finally, *web integration* was ranked highest in the evaluation as only very limited set of proprietary technologies or APIs were used while implementing the Feed Widget.

# Chapter 7

# Discussion

Chapter 6 evaluated the concept implementation against the key requirements specified in Section 2.5. The implementation satisfied most of the requirements. However, some requirements were found to be impractical or merely impossible to fulfill with the given platform using core web technologies. Regardless, experimental features of the web technologies implemented and tested in the development environment addressed many of those issues. In this chapter, the results of the evaluation are discussed with a specific focus on the main challenges confronted. In addition, the feasibility of alternative approaches suggested by the emerging web technologies which address the requirements are discussed.

## 7.1 Functionality

All the functional requirements set for the concept implementation defined in Section 5.1.3 were fulfilled. However, certain challenges were confronted during the implementation task. This section discusses main generic challenges and rationale for approaches taken.

### 7.1.1 Incompatible Browser and Rendering Engine Implementations

Incompatibility with and disregard toward web standards in web browsers has been a major issue since the emergence of multiple competing web browsers during the browser wars[1] in the late 90s. In the most common cases this leads to inconsistently rendered UI elements across user agents. However, while executing program logic within the browser, such incompatibilities may have more severe consequences which cause the application to fail completely. The DOM standard has a particular role in web applications, yet it is one of the most inconsistently implemented standards across browsers. Another major cause of incompatibility are the CSS implementations across the major browsers. Furthermore, native support for more recent web technologies such as SVG and XForms (see Appendix B.1) is flaky or nonexistent, which forces developers targeting widest possible audience – such as the most developers of commercial web applications do – to use the least common denominator of web standards. This is commonly considered to be HTML 4.01, CSS Level 2 and DOM Level 2 with various non-standard adaptations to support the market leader, the Internet Explorer. However, using this set of standards for implementing interactive web applications is pushing the limits of these ten-year-old web technologies.

The rendering engine of the WRT, the S60WebKit, defines the baseline for the Feed Widget implementation. Its main limitations were incompatibilities in its DOM Level 2 support especially related to event types. Secondly, the CSS implementation had inconsistencies across S60WebKit and major desktop browsers. Regardless, interoperable implementation was realizable, but required considerable amount of debugging and testing across different browsers.

The currently fragmented browser market for mobile devices is converging towards a common rendering engine which is used in major mobile platforms as suggested by the developments discussed in Section 4.2.3. However, the rapid development of the web standards and the lack of commitment of all the stakeholders to work together according to the open source development model is leading to a new type of interoperability challenge. Whereas traditionally the fragmentation has occurred between the different rendering engines, today the same is increasingly happening within the same rendering engine. In practice, different versions with varying level of standards compliance exists as the code of the open source rendering engines is branched at various stages of development and extended with proprietary extensions. This implies that authors have to potentially support various incompatible web runtime environments in addition to different rendering engines.

[1]*Browser wars* refers to the competition for dominance between the browser vendors in the web browser marketplace.

### 7.1.2 Role of the Declarative Markup

Generally, the main obstacle of the (X)HTML is its inherent thin client architecture in which all the modifications to the user interface are done on the server. Client-side interaction is limited to the document loaded to the client and even the simplest request to the server requires the whole page content to be loaded from the server. Interaction available is limited to hyperlinks used for navigating within the page or to other pages, that is information retrieval, and using forms used for submitting information (either text of binary data) to the web server, that is information manipulation. This leads to an unresponsive user interface and low interactivity. Luckily (X)HTML allows inclusion of other languages which alleviate this. Still the (X)HTML is forming the mandatory legacy base with its restrictions.

Within the Feed Widget the XHTML had a role of bootstrapping the application, and providing hooks for the scripts to attach to in order to interact with the DOM. In other words, XHTML was the root language used. UI was completely based on standard XHTML elements providing hooks to implement interaction. Beyond simple layouts and declaring the initial state, the program logic was entirely implemented in JavaScript and any modifications to the document were reflected to the DOM in runtime. The main deficiency of this approach in constructing the view was deemed to be the large amount of procedural JavaScript code required. Consequently, associated high reliance on DOM traversal introduced performance problems. In addition, the extensive use of JavaScript inherently lowers the semantic level of the implementation and makes designing for device independence more challenging. Extending declarative markup with UI elements which are more applicable to application scenarios similar to XUL and leveraging more efficient mechanism for traversing the DOM such as XPath [35] language would arguably limit the amount of procedural code, enhance the performance and make web applications easier to author in general.

Client-side web application frameworks discussed in Section 4.2.3 raise the semantic level by providing high-level JavaScript APIs for repetitive tasks in the domain of web application development. However, they do not degrade gracefully to less capable devices and do not interoperate with authoring tools particularly well. In addition, all tested frameworks introduced problems with the WRT, probably due to the fact that they were especially tailored for each major browser to overcome their incompatibilities without a focus on the specific needs of the S60WebKit. Other web runtime environments discussed in Section 4.2.4 typically incorporated more expressive XML-based declarative languages for declaring UIs, but were mostly not designed with device independence as the topmost priority as target platforms were typically fairly homogenous. However, it is expected that this will change as more web runtime environments start migrating to mobile devices and the value of seamless functionality of the applications without porting across desktops and mobile devices is realized.

### 7.1.3 Core JavaScript Limitations

Lessons learned from Java have shown that running everywhere, securely and interactively is hard to achieve. Judging by the fact that JavaScript is used in over half of all the web pages it seems that it has managed to solve some problems Java applets[2] faced earlier and is truly a cross-platform language [84, 110]. In addition, JavaScript is being increasingly used for more serious application development and is no longer considered a toy language applicable for fancy UI effects only. Today, a typical web browser executes orders of magnitude more JavaScript than Java, both in code volume as well as in dynamic instructions which demonstrates the ubiquitous availability of JavaScript interpreters in the web browsers [31].

However there are certain deficiencies in JavaScript. First, it lacks a mechanism for extending the language with libraries in an interoperable manner. This functionality is commonplace in popular server-side languages such Java, PHP or Python whose core language capabilities are commonly extended with libraries. Furthermore, the lack of a proper namespace mechanism hinders combining custom libraries arbitrarily. Due to this, e.g. handling non-native data formats in core JavaScript language required extra care and was deemed error-prone and tedious. On this account, JavaScript built in data formats were utilized whenever feasible. However, for dealing with XML formats non-standard components for XML parsing and DOM serialization were leveraged due to the absence of standard approach.

Better support for XML and associated technologies in JavaScript would have simplified the process of dealing with XML in Feed Widget and most probably provided better performance as well. However, currently support for technologies which facilitate dealing with XML in JavaScript, e.g. E4X, XSL Transformations and XPath is limited. Consequently, several somewhat incompatible implementations for browsers exists in addition to some JavaScript library implementations which by design offer subpar performance for most common tasks.

[2]Java applets provide a mechanism to execute Java bytecode in a web browser using a user-installable Java Virtual Machine (JVM) plugin.

Due to faint support for XML in current JavaScript implementations, some authors have recently embraced the JavaScript Object Notation (JSON) (see Section 3.6.4) as an XML replacement. JSON in an object-oriented data format which is natively supported by JavaScript. However, JSON lacks powerful querying capabilities similar to XPath and has less support in the development tools in general. Due to the fact that the data interchange formats consumed by the Feed Widget were all XML-based, JSON was not considered an alternative. However, in cases where the developer is in control of the server-side implementation, JSON provides a feasible alternative to XML. In addition, there has been a trend in open service APIs discussed in Section 4.1.2 toward the use of JSON over XML.

### 7.1.4 Data Format Incompatibilities

Related to specific data formats used in the Feed Widget some problems were faced. Namely, it was realized that the Feed Widget would need to support various RSS versions wild in the Web – out of which many did not strictly adhere to any of the specifications. To be exact, there exists nine different versions of RSS with different incompatible schemas in addition to Atom Syndication Format [155]. The vast amount of RSS schemas limited the feasible parser implementations to less elegant approach which does not validate the feeds against their schemas to limit the complexity of the implementation. It is also remarkable that such parsing functionality on the client-side must be implemented in JavaScript due to lack of exposure of more robust XML parser libraries of the S60 platform to the JavaScript such as the libxml2[3]. There existed no freely available parser implementation targeted for feed parsing in JavaScript, which forced one to be implemented from the scratch as a part of the concept implementation.

Another noteworthy compatibility problem with the feed formats, and RSS in particular, was related to embedded HTML markup. The Userland's RSS reader which is generally considered as the reference implementation did no filter the HTML markup on the feeds. Due to this, publishers polluted the feed payload with HTML markup and the reader applications of today are expected to cope with this unspecified behavior. The Feed Widget takes a liberal approach and strips all the HTML tags from the message payload in the feeds. The HTML content within feeds is commonly tag soup[4] and

[3]libxml2 is a well-known library for parsing XML documents implemented in C.

[4]Tag soup refers to HTML code written for a web page without regard for the rules of HTML structure and semantics.

parsing it into a DOM representation was deemed computationally too heavy task. Sanitizing the input also reduced the amount of data transferred as references to other resources such as images were removed.

All in all, the above implies that incompatibilities related to feed formats continue to exist despite more robust standards, and parsers will need to adapt to the situation for many years to come impeding interoperability.

### 7.1.5 Styling and Layout Issues

The standard way to define presentation of web content is CSS. It employs style rules which are declarative as they declare the desired style instead of the implementation details. It is the layout engine that interprets the CSS style rules and is completely responsible for the cascading process. Due to this, CSS was deemed fairly easy to author. However, as the CSS lacks concepts familiar from widget toolkits such as containers, windows and panes, authoring application UIs and especially defining scalable layouts was not straightforward. Additionally, some browser incompatibilities forced to employ unintuitive hacks to achieve desired functionality. In the worst case scenario, different CSS needs to be served to each browser which undermines the utility of the CSS over proprietary approaches. Recently, CSS has been used for defining temporal interaction as well, whereas traditionally, the task of CSS has been to interpret the declarative rules and transform them into a static visual representation only. This trend poses further demands on the CSS engine implementations and closes the gap between the graphics capabilities of proprietary solutions such as XAML and standards-based CSS. It was deemed that the CSS is the most potential candidate for a standards compliant device agnostic declarative presentation technology, and making the CSS implementations as rich and interoperable as possible was seen as a prime importance in order to limit the use of proprietary technologies for implementing presentational aspects of web applications.

CSS was used practically for defining all the presentation and style aspects of the Feed Widget. WRT API provided only limited complementary functionality comprising of a hardware accelerated fading transition. Since the rendering engine of the WRT – the S60WebKit – lacks support for SVG, no standards-based mechanism for scalable graphics was available. However it was deemed that the lack of scalability could be compensated with CSS features such as relative positioning to some extend. In general, the main issues with respect to the styling and layout aspects relate either to the broken CSS implementations of the rendering engines or to the limitations of the CSS specification itself. Inconsistent CSS implementations across browsers made designing pixel precise layouts impossible whereas most language limitations could be worked around using scripting, although this required more code and sacrificed some performance. The most severe limitation of the CSS Level 2 was its rather simple selector mechanism which does not support selecting ancestors or parents of matched element[5].

## 7.2 Efficiency

In this section the main issues related to the efficiency of the WRT and associated web technologies are discussed. The effects of the network and hardware on the efficiency are intentionally left out. An extensive study covering such factors on the S60 can be found in [153].

### 7.2.1 Core JavaScript Performance

The lack of a threading model in JavaScript [70, p. 255] meant that the document parsing stopped while code was loaded and executed. To alleviate this the Feed Widget broke the execution of computationally heavy calculations to discrete subtasks. Similarly, asynchronous HTTP requests were used to simulate multi-threading which effectively kept the application responsive while communication with the server over HTTP. A more robust albeit non-standard solution for more proper multi-threading is provided by the Google Gears (see Section 3.6.3) and its worker pool implementation which runs JavaScript code by another JavaScript interpreter instance in the background.

### 7.2.2 The DOM as a Global Data Model

By default, the modifications to the DOM exist only during the runtime of a web page. On this account, it is commonplace that to maintain the transient client-side state across views in interactive web applications, the DOM is kept intact and only those views not visible to the user are set as hidden but not removed. As a consequence, the DOM may grow in complexity which severely affects the performance of all the functionality tied to it. This behavior was also observed with the Feed Widget implementation.

[5]CSS Level 3 Selectors module is fixing the issue as defined in [82, chapter 8], however the specification is still a Working Draft and thus has very limited support in browsers.

DOM Level 3 Load and Save [184] defines a standard mechanism for loading XML document content into a DOM and serializing the DOM into an XML document which together with a client-side persistence mechanism enables storing DOM documents to the client-side. This could be leveraged to unload unused parts from the DOM to limit its size which should alleviate the above-mentioned issue with the performance.

### 7.2.3 Limited Support for State Management

Standards-based mechanisms for implementing state management include a data frame[6] [54, p. 390], HTTP cookie and server-side storage via asynchronous HTTP[7]. The only cross-browser cross-platform mechanism appropriate for persisting such data over a session is the HTTP cookie. However, cookies had severe deficiencies from the viewpoint of the Feed Widget. Although the cookie specification did not define any size limit for cookies, practically all current browsers limited the size and the number of cookies [70, p. 460]. Furthermore, cookies caused overhead to the HTTP client-server communication[8]. Furthermore, the API for interacting with cookies was deemed clumsy and error-prone as it exposed low-level details not relevant for most developers. The most severe flaw was that the API did not provide information whether the storing of a data into a cookie succeeded or not.

### 7.2.4 Various Client-side Data Persistence Mechanisms

Client-side persistence mechanism for resources fetched from the network enhances efficiency especially in mobile devices which typically communicate over slower and more costly cellular networks. WRT provides a simple API via `widget` object methods `setPreferenceForKey()` and `preferenceForKey()` exposed to JavaScript for storing data persistently to the client-side as key value pairs which was used for implementing offline functionality of the Feed Widget. The mechanism is applicable only for storing simple datasets consisting of text. It was found out that in order to store

[6]Data frame approach exploits the fact that modern browsers support frameset allowing a hidden frame to be used as a data repository.

[7]This can be implemented in the background using XMLHttpRequest API which allows the web browser scripting language such as JavaScript to communicate with a web server using HTTP asynchronously.

[8]The cookie data is not automatically removed from the HTTP message payload after use.

other than text-based data such as images, the data to be stored must be first encoded using text-based encoding scheme such as Base64 which presumably degrades the performance.

HTML 5 defines a client-side persistent storage which supports both simple key and value pairs as well an SQL database interface. In addition a complete caching mechanism for resources called *application caches* has been defined in [98, section 4.6.2]. Google Gears provides similar mechanism as a non-standard plugin extending the web browser with a relational database API. Many available approaches imply that a widely supported standard method for client-side storage is in demand and many practical use cases exists. The HTML 5 storage mechanism is already implemented by the latest versions of Firefox and Safari browsers which implies that offline functionality for web applications is finally becoming a reality.

In the Feed Widget, storing of resources was implemented by the `Storage` class introduced in Section 5.1.5. The data was persistently stored on the client-side using the most appropriate persistence mechanism available on the the given platform. The resources stored were identified by their URIs. Implementing proper support for versioning of stored resources was deemed a tedious, albeit not impossible task, with only a simple key value pair mapping.

### 7.2.5 Inefficient DOM and CSS Implementations

Significant components contributing to the overall performance, in addition to pure JavaScript execution speed, are DOM and CSS implementations of the rendering engine. As the DOM is increasingly used as a generic interface and extended with proper extensions, it is also becoming one of the main bottlenecks in the performance of the web applications developed in JavaScript. For example, in order to animate the movement of an element, its `style` property defining its position on the screen needs to be continuously updated using JavaScript to simulate animation. At the same time the layout engine of the browser reads the new position from the DOM and updates the view accordingly. This I/O model commonly used in web applications is considered complex, inefficient and clumsy [141, p. 7]. A partial solution to this problem exists in a form of `canvas` element of HTML 5 providing direct mode rendering. Alternatively, hardware accelerated graphics rendering of the most common temporal interactions defined in CSS Level 3 Animation, Transitions and Transforms modules (see Section 3.5.2) did introduce significant performance gains using the WebKit on the desktop.

However, measuring the performance improvement was deemed out of the scope of this thesis as there existed no hooks to which timers could have been attached for measurement without modifying the rendering engine of the WebKit.

Significant part of the performance equation is the consequence of the style-related DOM manipulations, the laying out process, which takes most of the time of the rendering engine. It includes tasks such as calculating the sizes of the visible elements and their positions in accordance with the cascading mechanism of the CSS. The research done by Pohja and Vuorimaa in [158] around the X-Smiles browser [163] indicates that the time it takes to layout the elements increases almost linearly as the size of the DOM grows, whereas the time it takes to paint such elements happens almost in constant time. This implies that the complexity of the DOM is the main cause for the performance drop in the rendering of the Feed Widget UI in cases in which multiple feeds have been loaded and their layout is modified.

## 7.3 Portability

### 7.3.1 Manifest Limitations

There are certain limitations in the existing WRT manifest format discussed in Section 4.2.5. First, the existing collection of metadata fields is fairly limited. Secondly, these metadata properties are not exposed to the JavaScript, they are only utilized by the S60 while the widget is installed or started. However, due to extensible nature of the XML, it should be fairly straightforward task to extend the manifest format. Furthermore, there exists no standard dialect for such manifest files due to the fact that the the specification [120] defining such properties has not yet been finalized which limits the interoperability.

### 7.3.2 Accessibility

Although the Web is increasingly used for creating interactive web content and web applications, today's core web technologies do not sufficiently support authoring accessible dynamic content for the Web. The main constraint is that accessibility of established web standards relies on abstracting semantics from both the content and the presentation information, typically by extracting semantic cues from (X)HTML tag element names [173]. However, this is not sufficient for contemporary web applications for two main reasons. Firstly, they use scripting to modify the DOM for creating dynamic custom UI components and secondly, any element – not just form controls such as buttons – allow attaching events to them which should be accessible with various input methods. Implications of this were most visible in the WRT related to its navigations model in which only links and form components gained focus in the focus navigation mode by default.

### 7.3.3 Security Sandbox and the Same Origin Policy

JavaScript is run in a security sandbox which prevents scripts from accessing multiple domains or performing other than web-related actions. This is enforced by the same origin policy[9] which is a de facto security measure implemented in web browsers. Related to interaction with web server, the policy prevents client-side scripts such as JavaScript loaded from one domain to interact with other domains which makes designing client-side application in JavaScript impossible if access to multiple domains is needed and no server-side proxy can be used. In addition, the policy restricts the access to host platform capabilities such as to local filesystem to read or write files. The rationale behind the policy is to not trust content loaded from any web site.

Above rationalizes the security model in which access to local resources is also restricted from within the browser. The above-mentioned limitations would have made designing Feed Widget according to its functional requirements impossible. Luckily, the policy was not enforced by the WRT and was possible to bypass for most browsers[10] while bootstrapping the application from the local filesystem. This relaxes the security policy by considering applications trusted. The trust is based on the user's decision to manually install such application prior to its use. This mechanism delegates the responsibility of trusting the application to the user in a same way as with any other user-installable application not running within security sandbox. However, the security mechanism should be revised in the coming releases of the WRT which expose more diverse set of platform APIs which may access sensitive data stored on the device.

A W3C Working Draft *Access Control for Cross-site Requests* [119] is defining a mechanism for interactive web applications for extending the same origin

[9]The same origin policy dates back to Netscape Navigator 2.0. `http://www.mozilla.org/projects/security/components/same-origin.html`

[10]Mozilla Firefox required minor alterations to its configuration to allow local resources to access multiple domains.

policy which allows web applications to interact with multiple domains. However, this needs support from web browsers, which may be challenged due to the fact that the latest version of the Internet Explorer has introduced its own incompatible model for cross-site requests[11]. Implication to portability is that currently proxy-based approach is the only truly portable solution up until the user agents have agreed on a standard way for cross-site request.

### 7.3.4 Device-independence and Multimodality

The device-independence principle encompasses all technologies and thus the Feed Widget was designed with that in mind on multiple levels. The following measures in the Feed Widget were taken. A fallback mechanism was always provided if non-standard JavaScript APIs were used. The built-in device-independence mechanism of the CSS allowing presentation to degrade gracefully if particular features were not supported was utilized. This included the hardware accelerated animations, transitions and transforms provided by the experimental CSS Level 3 modules. In terms of the multimodality, three alternative navigation mechanism were supported. It should be noted that the more advanced multimodality capabilities such as support for voice were not yet supported in mainstream browsers. However, such functionality can be expected to be of interest in the coming years due to the ubiquity of the Web and popularity of mobile devices.

### 7.3.5 Hypertext Navigation

Hypertext navigation refers to a traditional way of navigating across web pages using a web browser. To update page content, web applications utilizing Ajax technique update page fragments instead of requesting for a new web page identified by a new URI from the server. As a result, the state changes are not pushed to the navigation history of the browser. Furthermore, bookmarking of such page state is not possible. These problems originate from the fact that without changes in the URI of the resource, it still refers to the original state and neglects the subsequent changes done to the DOM and its visual representation, the UI. To overcome this limitation, the concept implementation builds on the default WRT navigation mechanism which prevents the user from using the navigation controls (e.g. back and forward navigation) and allows potentially any DOM element to be used as a

[11]Internet Explorer 8 introduced a proprietary Cross Domain Request object in March 2008 and it is unknown whether the mechanism will be standardized in W3C as of writing.

navigation control. However, this approach introduces obvious accessibility shortcomings as moving back and forth in application views is not possible programmatically.

### 7.3.6 Graceful Degradation

Various methods for implementing gracefully degrading web content was explored. As there is no formal way to deduce supported technologies based on the browser version programmatically, authors have to test the support for each feature prior use which is tedious and error prone. To test for DOM conformance by DOM modules, `hasFeature()` method can be used [70, p. 314]. Each feature exposed to JavaScript as an object, such as the support for XSLT, can be tested by object detection. However, as such object names are not standardized this is both error prone and inefficient. Due to the above-mentioned inconveniences, the concept implementation was build on web technologies inherently compliant with graceful degradation principles to the greatest extend. Presentation was entirely defined in CSS which adheres to graceful degradation principles – it silently discards unsupported rules such as the experimental CSS Transitions features described in Section 3.5.2.

The graceful degradation compliance was tested in practice as the development was done on a desktop with different browser engines[12] supporting more recent set of web standards in addition to the mobile device.

### 7.3.7 Modality Issues Related to the Event Model

The standard event types defined by `HTMLEvent` module were desktop browser and web page centric and did not particularly address the needs of mobile device input methods or the needs of more interactive applications. For example, the `input` element used to represent a text field in XHTML, did not support listening to events of `change` type in the WRT. This prevented implementing free-text search with auto-completion. Another issue arose with the drag and drop functionality which is a common interaction pattern implemented using `mousedown`, `mousemove` and `mouseup` event types defined in `MouseEvent` module. The problem was that those event types were not support by the WRT.

Key events are an important part of web applications which mimic the be-

[12]Gecko 1.8.1 and WebKit late November 2007 build.

havior of desktop application. Surprisingly, key events worked in the WRT although they are not formally standardized by the W3C [70, p. 425]. It was deemed that a cross-platform and cross-browser compatible event handling mechanism such as the one implemented by `EventDispatcher` of the concept implementation facilitated development considerably by providing an adaptation layer handling all the user generated events and hiding the incompatibilities from the developer.

## 7.4 Ease of Authoring

Development tools play a significant role in the application development. For web development, they range from modern integrated development environments to simple text editors. Regardless, there is currently no widely accepted theoretical foundation or well-established conceptual model for web application development [142, p. 6]. This implies that there are multiple ways to handle the development. As the development environment has a significant impact on the ease of authoring, the development approach of the Feed Widget is briefly discussed.

JavaScript is interpreted, that is the code is not compiled prior to its execution. Due to this inherent feature of the language, the evolutionary development approach [141, p. 9] was taken and the code was tested incrementally using the real platform during the development. The main development platform was a Firefox browser with a Firebug debugger plugin which facilitated the development by e.g. allowing runtime modifications to the DOM. Experimental features, such as CSS animation, transitions and transforms, were tested on a recent build of the WebKit. For testing on the actual platform, a beta version of the S60 Platform SDK for 3rd Edition Feature Pack 2 [49] was used to emulate the hardware device. Finally, in order to test the functionality on a real device, Nokia N95 was used.

The foremost challenge in development environment described above was the lack of proper tools for debugging on the emulator or on the actual device. For example, there were no facilities for defining breakpoints to inspect the state of the application during its execution in the emulator. Fortunately, the development and debugging of WRT applications on the above-mentioned desktop browser was possible. The WRT specific JavaScript objects described in Chapter 4.2.5, allowing the use of native device menus and persistent storage, were emulated in JavaScript [198].

The core of the WRT is the S60WebKit rendering engine, which is a port of

WebKit to the S60. Due to the fact that the S60WebKit uses an older release of WebKit than the latest version of Safari used for testing experimental CSS features, replicating similar host environment without resorting to emulator was not possible. Furthermore, there were differences in the standards support between Firefox and WebKit[13], especially related to support for defined and implicit animations, transforms and XML data processing.

## 7.5 Web Integration

Novel web technologies should integrate with the existing stack of technologies and protocols and re-use them when possible. This includes especially the base web technologies such as Uniform Resource Identifier (URI), Hypertext Transfer Protocol (HTTP) and Extensible Markup Language (XML) and other languages derived from it. These technologies were embraced in practically every platform discussed in this thesis, regardless, most of them introduced proprietary solutions on top of these fundamental base technologies instead of leveraging standards-based alternatives which impedes the integration opportunities.

The concept implementation did adhere to the REST principles discussed in Chapter 3.1. The client-server constraint was fulfilled, albeit the server component served only as a simple data provider. The statelessness principle was not infringed, as the persistence mechanism did not require the server to recall client states and no such state information was transmitted to the server. On this account, the cacheability principle was followed. Furthermore, the interface uniformity constraint was obeyed as only standard HTTP methods were used for client-server communications. As no new communication protocols were introduced in addition to HTTP, the constraint for layered system was conformed. Finally, the optional code-on-demand constraint was obeyed. Although the JavaScript was heavily used as a basis of executable components of the application, such program code was not delivered over HTTP.

[13]The late November 2007 build of WebKit was used for testing. The latest versions of the WebKit are available at `http://nightly.webkit.org/`.

## 7.6 Key Findings and Recommendations

This section provides a summary of the issues and observations related to the concept implementation and gives recommendations for corrective actions.

### 7.6.1 Language Issues

Issues stemming from the limitations of the the core markup and programming languages used restrict the functionality of the applications and hinder the development. The main factors are limited expressivity of declarative web technologies (XHTML, CSS) and functionality of the main procedural language, the JavaScript. Fortunately, the recent versions of the standards, namely HTML 5 and CSS Level 3, propose significant improvements maintaining backwards compatibility, and it is recommended that they are implemented by the user agents as the specifications mature.

### 7.6.2 Performance Issues

The performance issues of the languages dependent on the DOM, especially for advanced graphics effects, are common due to the fact that the core web technologies are re-used in tasks they were not originally designed for. For example, it is common to use the DOM as a global data model or combined with JavaScript leverage it to generate rich graphics effects. Recent versions of standards propose more optimal approaches. In addition, the closer integration of rendering engines with hardware-accelerated graphics engines should be studied as applications are increasingly adding richer graphical effects which are computationally heavy without hardware support.

### 7.6.3 Interoperability and Compatibility Issues

Problems with the interoperability between user agents limit the usability of web technologies due to which lowest common denominator of standards support is used if interoperability is of utmost importance. Especially, the lack of standard APIs for accessing device capabilities and a robust mechanism for deducing the set of supported web technologies and device characteristics impedes interoperability. To remedy this, vendors and standardization bodies should work together to make sure that standards are not formally finalized until relevant implementations exists. In an optimal but less realistic scenario, user agents would undergo a compatibility testing prior to their public release. Currently, such tests exist but compliance is entirely voluntary. Though it can be disputed, whether any formal body should be given such a mandate.

### 7.6.4 Usability and User Interaction Issues

Issues which relate to usability and user interaction mainly stem from the page-centric model of the Web which is not designed for application use cases. These issues have generally more serious implications on mobile devices which employ limited input and output mechanisms. The main issues observed related to the non-standard navigation mechanism, lack of support for multi-modality and the use of semantics not applicable for applications, especially related to events. As a solution, new technologies such as HTML 5 addressing the needs of applications are being developed. However, technologies can only partially address these issues and thus there is a need for developer guidelines targeted specifically to the problem domain.

### 7.6.5 Web Runtime Environment Issues

The main issues inherent to the web runtime environment, the S60 Web Runtime (WRT), are discussed herein. All the issues described in Sections 7.6 apply also to the WRT which inherits much of its functionality from the WebKit rendering engine. Generally, the main issue in user agents deployed in mobile devices has been their reliance on the operating system in a way that makes them hard or impossible to update independently of the entire software of the device. Specifically to the WRT and the S60, a mechanism to update the WebKit rendering engine component independent of the overall software update is missing. Secondly, there exist interoperability issues due to the use of non-standard platform-specific interfaces. To address this, an adaptation library was developed as part of the concept implementation. Alternatively, the interfaces could be aligned with those proposed in evolving standards to the extend feasible. Regardless, no standard currently specifies how to implement access to device capabilities using core web technologies which is the main functionality of the WRT.

# Chapter 8

# Conclusion

During the last 15 years or so, we have been witnessing the evolution of the Web usage where simple static web pages have been gradually gaining more and more features and functionality of applications traditionally implemented using device-dependent approaches. For most users today, the web browser encapsulates all the functionality needed to fulfill their computing needs and acts as an ubiquitous client to the applications and services. This blurs the visibility to and undermines the relevance of the underlying software layers such as operating systems both for the users as well as for most of the developers. In mobile context, however, this development has been impeded by the problems in using device-independent web technologies as building blocks of such applications and due to limited openness of mobile platforms in general. Recently, web technologies as building blocks of applications have been gaining momentum in mobile platforms as well.

In this thesis, the feasibility of client-side web technologies in mobile devices, and especially their applicability for implementing applications that are today typically developed using device dependent approaches was assessed. More specifically, the feasibility of a runtime platform for said applications, the S60 Web Runtime, was investigated.

To analyze the problem domain, requirements for typical web applications in mobile context were synthesized from the literature. Furthermore, core web technologies as well as platforms on top of which web applications are commonly built were discussed and compared. Furthermore, a taxonomy for web applications was drawn to illustrate the broad range of applications that are today commonly referred to by an umbrella term, web application. Similarly, platforms for web applications discussed in this thesis were categorized based on their level of abstraction and support for web technologies.

*Middleware* platforms typically bind lower-level languages to the Web by selectively leveraging key web standards, whereas *web runtime environments* commonly abstract lower-level details from developers and enable them to implement applications entirely in web technologies. Web browsers form a more established base for web applications built in core web technologies but they do not expose the functionality of the underlying platforms, and suffer from interoperability problems.

In the empirical part of this thesis, the concept implementation validated that it is feasible to develop applications which integrate to the existing web services as well as with the device capabilities using web technologies only in mobile devices. A rudimentary framework was developed for the S60 Web Runtime to ease the development of the concept implementation emphasizing the benefit for adding one more level of abstraction on top of typical core web technologies to facilitate application development. Potentially, leveraging web technologies opens the S60 platform to millions of new developers who master web technologies but are not familiar with lower-level languages. In addition, this should also remove the need to port applications across multiple platforms as it was found out that said application can be designed to be device-independent. However, the device-independence still requires considerable effort and level of expertise due to various incompatibilities of the platforms.

Working with more abstract, declarative languages in application development was deemed fairly convenient. Regardless, the concept implementation built around the S60 Web Runtime still required a great majority of the functionality to be implemented in procedural dynamic programming language, the JavaScript. It remains to be seen whether mixing two completely different language paradigms, declarative and procedural, is a feasible solution in the long run. To compete with the lower-level platforms, there is a need for more powerful animation capabilities, libraries, and interfaces to access device capabilities in web technologies. Emerging work in the area of client-side web technologies showed that the issues have been acknowledged and both standard and proprietary solutions are being worked on. The approaches of the proprietary offerings resemble much the embrace and extend strategy taken by major players during the browser wars in the 90s, in which a standard was adopted, but some non-standard features were added. To avoid market fragmentation and ensure long-term growth of the Web, the industry must agree on a set of open standards which form the base for future web applications.

To conclude, the research done in the domain of web applications and associated open web technologies around the client-side web technologies is still in flux. New standards are being developed, and at the same time proprietary solutions are targeting the same problem domain without embracing the existing standards. The increased performance of mobile devices is finally making the use of high-level cross-platform tools and web technologies a reality. However, the real challenge is to make web applications built on the aforesaid technologies truly interoperable across platforms for the benefit of the end-user. To realize this, the stakeholders have to work together instead of creating their own technological and political walled gardens.

## 8.1 Future Work

The domain of web applications and related web technologies is no doubt a hot topic. Despite the popularity of web applications, guidelines for web application developers are virtually non-existent, especially related to more restricted mobile devices. A more thorough study and synthesis of such guidelines would be of great practical value to developers.

The concept implementation revealed various usability and user interaction issues. Most of the issues stemmed from the page-centric model of the Web and associated technologies not directly applicable for realizing application use cases. Conducting a usability evaluation to assess the usability of applications built on various web runtime environments would give valuable information of the problems users face today.

In this thesis, the main bottlenecks of the S60 Web Runtime were discussed. However, a comprehensive study quantifying the results should be conducted to identify and optimize the exact components of the overall architecture which are critical to the performance.

There exists no standard way to access the device capabilities in an interoperable manner. On this account, a study comparing various approaches available for exposing platform functionality to web applications in a secure way provides a topical research subject.

Finally, there exists a plethora of client-side web application frameworks which are popular among web application developers but which fail to function properly on most mobile devices. Identifying and fixing the issues in both such frameworks as well as in user agents would considerably ease the development of cross-platform web applications.

# Bibliography

[1] Ajax Toolkit Survey, 2007. `http://ajaxian.com/archives/2007-ajax-tools-usage-survey-results`.

[2] M. Altheim, F. Boumphrey, S. Dooley, S. McCarron, S. Schnitzenbaumer, and T. Wugofski. Modularization of XHTML$^{TM}$. W3C Recommendation, 2001. `http://www.w3.org/TR/2001/REC-xhtml-modularization-20010410/`.

[3] M. Altheim and S. McCarron. XHTML$^{TM}$1.1 – Module-based XHTML. W3C Recommendation, 2001. `http://www.w3.org/TR/2001/REC-xhtml11-20010531/`.

[4] D. Appelquist, T. Mehrvarz, and A. Quint. Compound Document by Reference Use Cases and Requirements Version 1.0. W3C Working Draft, 2005. `http://www.w3.org/TR/2005/NOTE-CDRReqs-20051219/`.

[5] J. Axelsson, B. Epperson, Ishikawa M., S. McCarron, A. Navarro, and S. Pemberton. XHTML$^{TM}$2.0. W3C Working Draft, 2003. `http://www.w3.org/TR/2003/WD-xhtml2-20030506/`.

[6] Sullivan. B. Mobile Web Best Practices 2.0. W3C Editor's Draft, 2008. `http://www.w3.org/2005/MWI/BPWG/Group/Drafts/BestPractices-2.0/ED-mobile-bp2-20080213`.

[7] M. Baker, M. Ishikawa, S. Matsui, P. Stark, T. Wugofski, and T. Yamakami. XHTML$^{TM}$Basic. W3C Recommendation, 2000. `http://www.w3.org/TR/2000/REC-xhtml-basic-20001219/`.

[8] A. Barstow. Declarative Formats for Applications and User Interfaces. W3C Working Group Note. `http://www.w3.org/TR/2007/NOTE-dfaui-20070912/`.

[9] T. Berners-Lee. Information Management: A Proposal, 1989. http://www.w3.org/History/1989/proposal.html.

[10] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifier (URI): Generic Syntax, 2005. http://tools.ietf.org/html/rfc3986.

[11] T. Berners-Lee, L. Masinter, and M. McCahill. Uniform Resource Locators (URL), 1994. http://tools.ietf.org/html/rfc1738.

[12] P Bernstein. Middleware: A Model for Distributed System Services. *Communications of the ACM*, 39(2):86–98, 1996.

[13] E. Bertini, M. Billi, L. Burzagli, T. Catarci, F. Gabbanini, P. Graziani, S. Kimani, E. Palchetti, and G. Santucci. Usability and Accessibility in Mobile Computing Computing, 2004. http://www.mais-project.it/documenti_pubblico/IIIsemester/r7.3.3.pdf.

[14] M. Birbeck, S. McCarron, S. Pemberton, T. V. Raman, and R. Schwerdtfeger. XHTML Role Attribute Module, 2006. http://www.w3.org/TR/2006/WD-xhtml-role-20061113/.

[15] J.O. Borchers. A Pattern Approach to Interaction Design. *AI & Society*, 15(4):359–376, 2001.

[16] B. Bos. An Essay on W3C's Design Principles, 2003. http://www.w3.org/People/Bos/DesignGuide/introduction.

[17] B. Bos, T. Celik, I. Hickson, and W. Hakon. Cascading Style Sheets Level 2 Revision 1 (CSS 2.1). W3C Candidate Recommendation, 2007. http://www.w3.org/TR/2007/CR-CSS21-20070719/.

[18] B. Bos, H. Lie, C. Lilley, and I. Jacobs. Cascading Style Sheets Level 2. W3C Recommendation, 1998. http://www.w3.org/TR/1998/REC-CSS2-19980512/.

[19] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. Nielsen, S. Thatte, and D. Winer. Simple Object Access Protocol (SOAP) 1.1. W3C Note, 2000. http://www.w3.org/TR/2000/NOTE-SOAP-20000508/.

[20] T. Bray, J. Paoli, C. Sperberg-McQueen, E. Maler, F. Yergeau, and J. Cowan. Extensible Markup Language (XML) 1.1 (Second Edition). W3C Recommendation, 2006. http://www.w3.org/TR/2006/REC-xml11-20060816/.

[21] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible Markup Language (XML) 1.0 (Fourth Edition). W3C Recommendation, 2006. `http://www.w3.org/TR/2006/REC-xml-20060816/`.

[22] Millward Brown. Browser Plug-in Technology Study. `http://www.adobe.com/products/player_census/methodology/`.

[23] Browser Statistics, November 2007. `http://www.thecounter.com/stats/2007/November/browser.php`.

[24] I Bulterman, J. Jansen, S. Mullender, M. DeMeglio, J. Quint, P. Vuorimaa, S. Cruz-Lara, H. Kawamura, D. Weck, E. Hyche, X. Paneda, D. Melendi, T. Michel, and D. Zucker. Synchronized Multimedia Integration Language (SMIL 3.0). W3C Candidate Recommendation, 2008. `http://www.w3.org/TR/2008/CR-SMIL3-20080115/`.

[25] M. Caceres. Client-Side Web Applications (Widgets) Requirements. W3C Working Draft, 2006. `http://www.w3.org/TR/2006/WD-WAPF-REQ-20061109/`.

[26] M. Caceres. Widgets 1.0 Requirements. W3C Working Draft, 2007. `http://www.w3.org/TR/2007/WD-widgets-reqs-20070705/`.

[27] B. Caldwell, M. Cooper, L. Guarino, and G. Vanderheiden. Web Content Accessibility Guidelines 2.0, 2007.

[28] E. Candell and D. Raggett. Multimodal Interaction Use Cases. W3C Working Draft, 2002. `http://www.w3.org/TR/2002/NOTE-mmi-use-cases-20021204/`.

[29] M. Chambers, D. Dura, and K. Hoyt. *Adobe Integrated Runtime for JavaScript Developers*. O'Reilly, 2007.

[30] S. Champeon. Progressive Enhancement and the Future of Web Design, 2003. `http://www.webmonkey.com/03/21/index3a.html`.

[31] M. Cheng, M. Bebenita, A. Yermolovich, and A. Gal. Efficient Just-In-Time Execution of Dynamically Typed Languages Via Code Specialization Using Precise Runtime Type Inference. Technical report, Department of Computer Science, University of California, Irvine, 2007. `http://www.ics.uci.edu/~franz/Site/pubs-pdf/ICS-TR-07-10.pdf`.

[32] W. Chisholm, G. Vanderheiden, and I. Jacobs. Web Content Accessibility Guidelines 1.0. W3C Recommendation, 1999. `http://www.w3.org/TR/1999/WAI-WEBCONTENT-19990505/`.

[33] L. Chittaro and P. Dal Cin. Evaluating Interface Design Choices on WAP Phones: Navigation and Selection. *Personal and Ubiquitous Computing*, 6(4):237–244, 2002.

[34] A. Chuter. Relationship Between Mobile Web Best Practices 1.0 and Web Content Accessibility Guidelines. W3C Working Draft, 2008. `http://www.w3.org/TR/2008/WD-mwbp-wcag-20080122/`.

[35] J. Clark and S. DeRose. XML Path Language (XPath) Version 1.0. W3C Recommendation, 1999. `http://www.w3.org/TR/1999/REC-xpath-19991116`.

[36] M. Cokus and S. Pericas-Geertsen. XML Binary Characterization Use Cases. W3C Working Draft, 2007. `http://www.w3.org/TR/2005/WD-xbc-use-cases-20050224/`.

[37] comScore Inc. Mobile Phone Web Users Nearly Equal PC Based Internet Users in Japan, September 2007. `http://www.comscore.com/press/release.asp?press=1742`.

[38] Evans Data Corporation. Global Developer Population and Demographics Report, 2006.

[39] Microsoft Corporation. Fundamentals of Microsoft .NET Compact Framework Development for the Microsoft .NET Framework Developer, 2003. `http://msdn2.microsoft.com/en-us/library/Aa446549.aspx`.

[40] Microsoft Corporation. Windows Vista User Experience Guidelines, 2006. `http://msdn.microsoft.com/library/?url=/library/en-us/UxGuide/UXGuide/Home.asp`.

[41] Microsoft Corporation. innerHTML Property, 2007. `http://msdn2.microsoft.com/en-us/library/ms533897.aspx`.

[42] Microsoft Corporation. .NET Framework Programming, 2007. `http://msdn2.microsoft.com/en-us/library/ms229284.aspx`.

[43] Microsoft Corporation. How to: Use the WebBrowser Control in the .NET Compact Framework, 2008. `http://msdn2.microsoft.com/en-us/library/ms229657.aspx`.

[44] Nokia Corporation. Web Browser for S60. http://www.s60.com/browser.

[45] Nokia Corporation. MIDP: Scalable 2D Vector Graphics API Developer's Guide v1.1, 2006. http://www.forum.nokia.com/info/sw.nokia.com/id/3d0913bf-73f0-4aba-b1c3-754de783a6d3/MIDP_Scalable_2D_Vector_Graphics_API_Dev_Guide_v1_1_en.pdf.html.

[46] Nokia Corporation. Nokia N95 Device Details, 2006. http://www.forum.nokia.com/devices/N95.

[47] Nokia Corporation. Browser Control API Developer's Guide v2.0, 2007. http://www.forum.nokia.com/info/sw.nokia.com/id/47d8a7fe-768c-44e5-bc26-fcba0a05e35e/S60_Platform_Browser_Control_API_Guide_v2_0_en.pdf.html.

[48] Nokia Corporation. S60 Platform: Scalable UI Guideline, 2007. http://www.forum.nokia.com/info/sw.nokia.com/id/4239db2a-2e0d-4592-a9c0-3936d0550d64/S60_Platform_Scalable_UI_Guideline_v1_0_en.pdf.html.

[49] Nokia Corporation. S60 Platform SDK for 3rd Edition Feature Pack 2, Beta, 2007. http://www.forum.nokia.com/info/sw.nokia.com/id/4a7149a5-95a5-4726-913a-3c6f21eb65a5/S60-SDK-0616-3.0-mr.html.

[50] Nokia Corporation. S60WebKit, 2007. http://opensource.nokia.com/projects/S60browser/.

[51] Nokia Corporation. Web Runtime, 2007. http://www.s60.com/webruntime.

[52] Nokia Corporation. Web Runtime API Reference 1.1, 2007. http://www.forum.nokia.com/info/sw.nokia.com/id/b9ad2b23-07ea-46cd-badf-f0ba3df97da3/Web_Run_Time_API_Reference_v1_1_en.pdf.html.

[53] Nokia Corporation. Press Release: Nokia to Bring Microsoft Silverlight Powered Experiences to Millions of Mobile Users, 2008. http://www.nokia.com/A4136001?newsid=1197788.

[54] D. Crane, B. Bibeault, and J. Sonneveld. *Ajax in Practice.* Manning, 2007.

[55] R. Cremin and J. Rabin. dotMobi Switch On! Web Developer Guide, 2006. `http://dev.mobi/files/dotmobi_Switch_On_Web_Developer_Guide.html`.

[56] D. Crockford. The application/json Media Type for JavaScript Object Notation (JSON), 2006. `http://tools.ietf.org/html/rfc4627`.

[57] CSS3.info – Everything You Need to Know About CSS3, 2007. `http://www.css3.info/`.

[58] A. Danesh, K. Inkpen, F. Lau, K. Shu, and K. Booth. Geney$^{TM}$: Designing a Collaborative Activity for the Palm Handheld Computer. In *CHI '01: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 388–395, New York, NY, USA, 2001. ACM Press.

[59] I. David and M. Stachowiak. Window Object 1.0, 2006. `http://www.w3.org/TR/2006/WD-Window-20060407/`.

[60] J. Dean. Web API Working Group Charter, 2005. `http://www.w3.org/2006/webapi/admin/charter`.

[61] M. Dubinko, L. Klotz, R. Merrick, and T. Raman. XForms 1.0. W3C Recommendation, 2003. `http://www.w3.org/TR/2003/REC-xforms-20031014/`.

[62] M. Dubinko and T. Raman. XForms 1.0 Basic Profile. W3C Candidate Recommendation, 2003. `http://www.w3.org/TR/2003/CR-xforms-basic-20031014/`.

[63] ECMAScript Language Specification, Standard ECMA-262 3rd Edition, 1999. `http://www.ecma-international.org/publications/files/ecma-st/ECMA-262.pdf`.

[64] Proposed ECMAScript 4th Edition Ű Language Overview, 2007. `http://www.ecmascript.org/es4/spec/overview.pdf`.

[65] D. Fallside and P. Walmsley. XML Schema Part 0: Primer Second Edition. W3C Recommendation, 2004. `http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/`.

[66] M.E. Fayad, D.C. Schmidt, and R.E. Johnson. *Building Application Frameworks: Object-oriented Foundations of Framework Design.* John Wiley & Sons, Inc. New York, NY, USA, 1999.

[67] J. Ferraiolo, J. Fujisawa, and D. Jackson. Scalable Vector Graphics (SVG) 1.1 Specification. W3C Recommendation, 2003. `http://www.w3.org/TR/2003/REC-SVG11-20030114/`.

[68] R. Fielding. *Architectural Styles and the Design of Network-based Software Architectures.* PhD thesis, University of California, Irvine, 2000. `http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm`.

[69] R. Fielding, J. Gettys, J. Mogul, H. Nielsen, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1, 1999. `http://tools.ietf.org/html/rfc2616`.

[70] D. Flanagan. *JavaScript: The Definitive Guide.* O'Reilly, 2006.

[71] The WAP Forum. WAP 1.0 Specification Suite, 1998. `http://www.wapforum.org/what/technical_1_0.htm`.

[72] WAP Forum. WAP Architecture, 2001. `http://www.openmobilealliance.org/tech/affiliates/wap/wap-210-waparch-20010712-a.pdf`.

[73] WAP Forum. XHTML Mobile Profile, 2001. `http://www.openmobilealliance.org/tech/affiliates/wap/wap-277-xhtmlmp-20011029-a.pdf`.

[74] Mozilla Foundation. ARIA: Accessible Rich Internet Applications, 2007. `http://developer.mozilla.org/en/docs/Accessible_DHTML`.

[75] Mozilla Foundation. Core JavaScript 1.5 Reference, 2007. `http://developer.mozilla.org/en/docs/Core_JavaScript_1.5_Reference`.

[76] Mozilla Foundation. XULRunner, 2007. `http://developer.mozilla.org/en/docs/XULRunner`.

[77] Mozilla Foundation and Opera Software. Web Applications and Compound Documents. W3C Workshop Position Paper, 2004. `http://www.w3.org/2004/04/webapps-cdf-ws/papers/opera.html`.

[78] M. Fowler. GUI Architectures, 2006. `http://www.martinfowler.com/eaaDev/uiArchs.html`.

[79] J.J. Garrett. Ajax: A New Approach to Web Applications, 2005. `http://www.adaptivepath.com/publications/essays/archives/000385.php`.

[80] Gartner, Inc., Worldwide PDA and Smartphone Shipments Grow 26% in 1Q07, June 2007.

[81] R. Gimson, L. Suryanarayana, and M. Lauff. Core Presentation Characteristics: Requirements and Use Cases. W3C Working Draft, 2003. `http://www.w3.org/TR/2003/WD-cpc-req-20030510/`.

[82] D. Glazman, T. Celik, I. Hickson, P. Linss, and J. Williams. Media Queries. W3C Working Draft, 2005. `http://www.w3.org/TR/2005/WD-css3-selectors-20051215/`.

[83] C. Goldfarb. Information Processing – Text and Office Systems – Standard Generalized Markup Language (SGML), 1986.

[84] Google. Web Authoring Statistics, 2005. `http://code.google.com/webstats/`.

[85] Google. Google Web Toolkit, 2006. `http://code.google.com/webtoolkit/`.

[86] Google. Google Gears, 2007. `http://gears.google.com/`.

[87] Google. Google Reader, 2008. `http://reader.google.com/`.

[88] G. Grassel. Mobile Phonebook Mash-up Application Developed Using Web Technologies. In *XTech Conference*, 2007. `http://2007.xtech.org/public/schedule/detail/210`.

[89] Nielsen Norman Group. Nielsen Norman Group Papers and Publications, 2007. `http://www.nngroup.com/reports/`.

[90] A. Hagans. High Accessibility Is Effective Search Engine Optimization, 2005. `http://alistapart.com/articles/accessibilityseo`.

[91] S. Henry. Essential Components of Web Accessibility, 2005. `http://www.w3.org/WAI/gettingstarted/`.

[92] S. Henry. Accessible Rich Internet Applications (WAI-ARIA) Suite Overview, 2006. `http://www.w3.org/WAI/intro/aria.php`.

[93] I. Hickson. Sending XHTML as text/html Considered Harmful. `http://www.hixie.ch/advocacy/xhtml`.

[94] A. Holdener. *Ajax: The Definitive Guide.* O'Reilly, 2008.

[95] M. Honkala. *Web User Interaction – a Declarative Approach Based on XForms.* PhD thesis, Helsinki University of Technology, 2006. http://lib.tkk.fi/Diss/2007/isbn9789512285662/.

[96] A. Hors, P. Hégaret, L. Wood, G. Nicol, J. Robie, M. Champion, and S. Byrne. Document Object Model (DOM) Level 2 Core Specification. W3C Recommendation, 2000. http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113/.

[97] P. Hoschka. Mobile Web Initiative Activity Statement, 2007. http://www.w3.org/2005/MWI/Activity.

[98] HTML 5. W3C Working Draft, 2008. http://www.w3.org/TR/2008/WD-html5-20080122/.

[99] HTML 5 Differences from HTML 4. W3C Working Draft, 2008. http://www.w3.org/TR/2008/WD-html5-20080122/.

[100] HTML Design Principles. W3C Working Draft, 2007. http://www.w3.org/TR/2007/WD-html-design-principles-20071126/.

[101] D. Hyatt, D. Jackson, and C. Marrin. Apple's Proposal for CSS Animation, 2008. http://webkit.org/specs/CSSVisualEffects/CSSAnimation.html.

[102] D. Hyatt, D. Jackson, and C. Marrin. Apple's Proposal for CSS Transforms, 2008. http://webkit.org/specs/CSSVisualEffects/CSSTransforms.html.

[103] D. Hyatt, D. Jackson, and C. Marrin. Apple's Proposal for CSS Transitions, 2008. http://webkit.org/specs/CSSVisualEffects/CSSTransforms.html.

[104] IEEE. IEEE Std 1003.1-1988 Standard for Information Technology - Portable Operating System Interface (POSIX) - Base Definitions, 1988.

[105] Apple Inc. iPhone Safari. http://www.apple.com/iphone/internet/.

[106] Apple Inc. Dashboard Reference, 2006. http://developer.apple.com/documentation/AppleApplications/Reference/Dashboard_Ref/Dashboard_Ref.pdf.

[107] Apple Inc. Dashboard Programming Topics, 2007. `http://developer.apple.com/documentation/AppleApplications/Conceptual/Dashboard_ProgTopics/Dashboard_ProgTopics.pdf`.

[108] Apple Inc. Optimizing Web Applications and Content for iPhone, 2007. `http://developer.apple.com/webapps/designingcontent.php`.

[109] Apple Inc. iPhone Human Interface Guidelines for Web Applications, 2008. `http://developer.apple.com/documentation/iPhone/Conceptual/iPhoneHIG/iPhoneHIG.pdf`.

[110] E-Soft Inc. SecuritySpace Technology Penetration Report, 2007. `http://www.securityspace.com/s_survey/data/man.200708/techpen.html`.

[111] Ecma International. Standard ECMA-357: ECMAScript for XML (E4X) Specification, 2005. `http://www.ecma-international.org/publications/standards/Ecma-357.htm`.

[112] D. Jackson. Web Application Formats Working Group Charter. `http://www.w3.org/2006/appformats/admin/charter.html`.

[113] I. Jacobs. The W3C Technology Stack, 2007. `http://www.w3.org/Consortium/technology`.

[114] I. Jacobs, J. Gunderson, and E. Hansen. User Agent Accessibility Guidelines 1.0. W3C Recommendation, 2002. `http://www.w3.org/TR/2002/REC-UAAG10-20021217/`.

[115] I. Jacobs and N. Walsh. Architecture of the World Wide Web, Volume One, 2004.

[116] Christian Kaas. An Introduction to Widgets with Particular Emphasis on Mobile Widgets. Technical report, Hagem University of Applied Sciences, 2007. `http://www.symbianresources.com/tutorials/techreports/widgets/kaar07widgets.pdf`.

[117] E. Kaasinen, M. Aaltonen, J. Kolari, S. Melakoski, and T. Laakko. Two Approaches to Bringing Internet Services to WAP Devices. *Computer Networks*, 33(1-6):231–246, 2000.

[118] A. Kesteren. The XMLHttpRequest Object. W3C Working Draft, 2007. `http://www.w3.org/TR/2007/WD-XMLHttpRequest-20071026/`.

[119] A. Kesteren. Access Control for Cross-site Requests. W3C Working Draft, 2008. `http://www.w3.org/TR/2008/WD-access-control-20080214/`.

[120] A. Kesteren and M. Caceres. Widgets 1.0. W3C Working Draft, 2007. `http://www.w3.org/TR/2007/WD-widgets-20071013/`.

[121] A. Kivi. Mobile Browsers. *Helsinki University of Technology*, 2007.

[122] P.P. Koch. *PPK on JavaScript*. New Riders, 2007.

[123] D. Kristol and L. Montulli. HTTP State Management Mechanism, 2000. `http://tools.ietf.org/html/rfc2965`.

[124] S. Laakso. User Interface Design Patterns, 2003. `http://www.cs.helsinki.fi/u/salaakso/patterns`.

[125] A. Le Hors, G. Nicol, L. Wood, M. Champion, and S. Byrne. Document Object Model (DOM) Level 2 Core Specification. W3C Recommendation, 2000. `http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113/`.

[126] R. Lewis. Delivery Context Ontology. W3C Working Draft, 2007. `http://www.w3.org/TR/2007/WD-dcontology-20071221/`.

[127] H. Lie and B. Bos. Cascading Style Sheets Level 1. W3C Recommendation, 1996. `http://www.w3.org/TR/REC-CSS1-961217`.

[128] Linux to Be the Fastest-Growing Smartphone OS over the Next 5 Years, August 2007. `http://www.abiresearch.com/abiprdisplay.jsp?pressid=922`.

[129] Open Mobile Alliance Ltd. OMA Browsing Enabler Releases and Specifications, 2007. `http://www.openmobilealliance.org/release_program/browsing_archive.html`.

[130] K. Luchini, C. Quintana, J. Krajcik, C. Farah, N. Nandihalli, K. Reese, A. Wieczorek, and E. Soloway. Scaffolding in the Small: Designing Educational Supports for Concept Mapping on Handheld Computers. In *CHI '02: CHI '02 Extended Abstracts on Human Factors in Computing Systems*, pages 792–793, New York, NY, USA, 2002. ACM Press.

[131] Florins M. and J. Vanderdonckt. Graceful Degradation of User Interfaces as a Design Method for Multiplatform Systems. In *IUI '04: Proceedings of the 9th International Conference on Intelligent User Interfaces*, pages 140–147, New York, NY, USA, 2004. ACM Press.

[132] M. Mace. Growth of Web Applications in the US: Rapid Adoption, But Only When There's a Real Benefit, 2007. http://www.rubiconconsulting.com/thinking/whitepaper/2007/09/growth_of_web_applications_in.html.

[133] C. Mariage, J. Vanderdonckt, and C. Pribeanu. State of the Art of Web Usability Guidelines. *The Handbook of Human Factors in Web Design*, 2004.

[134] T. Mehrvarz, L. Pajunen, J. Quint, and D. Appelquist. WICD Core 1.0. W3C Candidate Recommendation, 2007. http://www.w3.org/TR/2007/CR-WICD-20070718/.

[135] T. Mehrvarz, L. Pajunen, J. Quint, and D. Appelquist. WICD Mobile 1.0. W3C Candidate Recommendation, 2007. http://www.w3.org/TR/2007/CR-WICDMobile-20070718/.

[136] A. Mesbah and A. van Deursen. An Architectural Style for Ajax. *WICSAŠ07: 6th Working IEEE/IFIP Conference on Software Architecture*, 2007.

[137] P. Metz, J. O'Brien, and W. Weber. Specifying Use Case Interaction: Types of Alternative Courses. *Journal of Object Technology*, 2(2):111–131, 2003.

[138] Microsoft. Silverlight Architecture, 2008. http://msdn2.microsoft.com/en-us/library/bb404713.aspx.

[139] Sun Microsystems. The K Virtual Machine (KVM), 2000. http://java.sun.com/products/cldc/wp/.

[140] Sun Microsystems. Java ME Technology, 2007. http://java.sun.com/javame/technology/.

[141] T. Mikkonen and A. Taivalsaari. Using JavaScript as a Real Programming Language. Technical report, Sun Microsystems Laboratories, 2007. http://research.sun.com/techrep/2007/smli_tr-2007-168.pdf.

[142] T. Mikkonen and A. Taivalsaari. Web Applications - Spaghetti Code for the 21th Century. Technical report, Sun Microsystems Laboratories, 2007. http://research.sun.com/techrep/2007/smli_tr-2007-166.pdf.

[143] T. Mikkonen, A. Taivalsaari, D. Ingalls, and K. Palacz. Web Browser as an Application Platform: The Lively Kernel Experience. Technical report, Sun Microsystems Laboratories, 2008. `http://research.sun.com/techrep/2008/smli_tr-2008-175.pdf`.

[144] E. Mitukiewicz and F. Telecom. Scope of Mobile Web Best Practices. W3C Working Group Note, 2005. `http://www.w3.org/TR/2005/NOTE-mobile-bp-scope-20051220/`.

[145] .NET Compact Framework for Symbian OS, 2007. `http://www.redfivelabs.com/`.

[146] M. Nicola and J. John. XML Parsing: A Threat to Database Performance. In *CIKM '03: Proceedings of the Twelfth International Conference on Information and Knowledge Management*, pages 175–178, New York, NY, USA, 2003. ACM.

[147] J. Nielsen. *Designing Web Usability: The Practice of Simplicity*. New Riders Publishing Thousand Oaks, CA, USA, 1999.

[148] J. Nielsen. Ephemeral Web-Based Applications, 2002. `http://www.useit.com/alertbox/20021125.html`.

[149] J. Nielsen. Why Ajax Sucks (Most of the Time), 2005. `http://www.usabilityviews.com/ajaxsucks.html`.

[150] Nokia Corporation. Nokia Web Browser Design Guide, 2007. `http://www.forum.nokia.com/info/sw.nokia.com/id/f8057b17-e5d7-43da-929f-34b33459d383/Nokia_Web_Browser_Design_Guide_v1_0_en.pdf.html`.

[151] Nokia Corporation. S60 Platform: Introductory Guide, 2008. `http://www.forum.nokia.com/info/sw.nokia.com/id/fc17242f-9bb2-4509-b12c-1e6b8206085b/S60_Platform_Introductory_Guide_v1_6_en.pdf.html`.

[152] M. Nottingham and R. Sayre. The Atom Syndication Format, 2005. `http://tools.ietf.org/html/rfc4287`.

[153] O. Ojala. *Service Oriented Architecture in Mobile Devices: Protocols and Tools*. Master's thesis, Helsinki University of Technology, 2006.

[154] S. Pemberton et al. XHTML$^{TM}$1.0: The Extensible HyperText Markup Language. W3C Recommendation, 2000. `http://www.w3.org/TR/2000/REC-xhtml1-20000126/`.

[155] M. Pilgrim. The Myth of RSS Compatibility, 2004. http://diveintomark.org/archives/2004/02/04/incompatible-rss.

[156] T. Pixley. Document Object Model (DOM) Level 2 Events Specification. W3C Recommendation, 2000. http://www.w3.org/TR/2000/REC-DOM-Level-2-Events-20001113/.

[157] M. Pohja, M. Honkala, M. Penttinen, P. Vuorimaa, and P. Ervamaa. Web User Interaction – Comparison of Declarative Approaches. In *Proceedings of the 2nd International Conference on Web Information Systems and Technologies (WEBIST 2006)*, pages 295–302, April 2006.

[158] M. Pohja and P. Vuorimaa. CSS Layout Engine for Compound Documents. *Third Latin American Web Congress (LA-WEB'2005)*, pages 148–156, 2005.

[159] A. Popescu, R. Geisler, E. Vartiainen, and G. Grassel. Mini Map - A Web Page Visualization Method for Mobile Phones. In *XTech Conference*, 2006. http://xtech06.usefulinc.com/schedule/paper/91.

[160] Java Community Process. JSR 226: Scalable 2D Vector Graphics API for J2ME, 2006. http://jcp.org/en/jsr/detail?id=226.

[161] Java Community Process. JSR 287: Scalable 2D Vector Graphics API 2.0 for Java ME, 2007. http://jcp.org/en/jsr/detail?id=287.

[162] Java Community Process. JSR 290: Java Language & XML User Interface Markup Integration, 2007. http://jcp.org/en/jsr/detail?id=290.

[163] X-Smiles Project. X-Smiles – An Open XML-browser for Exotic Devices, 2007. http://www.xsmiles.org/.

[164] L. Quin. The Extensible Stylesheet Language Family (XSL). W3C Recommendation, 2007. http://www.w3.org/Style/XSL/.

[165] A. Quint. Putting SVG and CDF to Use in an Internet Desktop Application. In *XTech Conference*, 2007. http://2007.xtech.org/public/schedule/detail/53.

[166] J. Rabin and C. McCathieNevile. Mobile Web Best Practices 1.0. W3C Recommendation, 2006. http://www.w3.org/TR/2006/PR-mobile-bp-20061102/.

[167] D. Raggett, A. Le Hors, and I. Jacobs. HTML 4.01 Specification. W3C Recommendation, 1999. `http://www.w3.org/TR/1999/REC-html401-19991224/`.

[168] J. Resig. getElementsByClassName Speed Comparison, 2007. `http://ejohn.org/blog/getelementsbyclassname-speed-comparison/`.

[169] L. Richardson, S. Ruby, and D. Heinemeier. *RESTful Web Services*. O'Reilly, 2007.

[170] V. Roto. Browsing on Mobile Phones. *Nokia Research Center*, 2005.

[171] V. Roto. *Web Browsing on Mobile Phones – Characteristics of User Experience*. PhD thesis, Helsinki University of Technology, 2006. `http://research.nokia.com/people/virpi_roto/dissertation.html`.

[172] V. Roto, A. Popescu, A. Koivisto, and E. Vartiainen. Minimap: A Web Page Visualization Method for Mobile Phones. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 35–44, 2006.

[173] R. Schwerdtfeger. Roadmap for Accessible Rich Internet Applications (WAI-ARIA Roadmap). W3C Working Draft, 2007. `http://www.w3.org/TR/2007/WD-aria-roadmap-20071019/`.

[174] L. Seeman and M. Cooper. States and Properties Module for Accessible Rich Internet Applications (WAI-ARIA States and Properties). W3C Working Draft, 2007. `http://www.w3.org/TR/2007/WD-aria-state-20070601/`.

[175] R. Simon, M. Kapsch, and F. Wegscheider. A Generic UIML Vocabulary for Device- and Modality Independent User Interfaces. In *WWW Alt. '04: Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers & Posters*, pages 434–435, New York, NY, USA, 2004. ACM Press.

[176] Opera Software. Opera Mobile. `http://www.opera.com/products/mobile/`.

[177] Opera Software. Opera Widgets Specification 1.0, 2007. `http://dev.opera.com/articles/view/opera-widgets-specification-1-0/`.

[178] C. Sorrel. Half the World Will Own Mobile Phones This Year, 2007. `http://blog.wired.com/gadgets/2007/06/half-the-world-.html`.

[179] M. Stachowiak. Understanding HTML, XML and XHTML, 2006. `http://webkit.org/blog/?p=68`.

[180] ISO Standard. TS 9241: Ergonomic Requirements for Office Work with Visual Display Terminals. *International Organization for Standardization, Geneva, Switzerland*, 1999.

[181] ISO Standard. TS 9126: Software Product Quality - Part 1: Quality Model. *International Organization for Standardization, Geneva, Switzerland*, 2001.

[182] ISO Standard. TS 16071: Ergonomics of Human-system Interaction – Guidance on Accessibility for Human-computer Interfaces. *International Organization for Standardization, Geneva, Switzerland*, 2003.

[183] J. Stenback, P. Hegaret, and A. Hors. Document Object Model (DOM) Level 2 Events Specification. W3C Recommendation, 2003. `http://www.w3.org/TR/2003/REC-DOM-Level-2-HTML-20030109/`.

[184] J. Stenback and A. Heninger. Document Object Model (DOM) Level 3 Load and Save Specification, 2004. `http://www.w3.org/TR/2004/REC-DOM-Level-3-LS-20040407/`.

[185] A. Swartz. RDF Site Summary (RSS) 1.0, 2000. `http://web.resource.org/rss/1.0/`.

[186] The Internet Corporation for Assigned Names and Numbers. MIME Media Types. `http://www.iana.org/assignments/media-types/`.

[187] J. Treviranus, C. McCathieNevile, I. Jacobs, and J. Richards. Authoring Tool Accessibility Guidelines 1.0. W3C Recommendation, 2000. `http://www.w3.org/TR/2000/REC-ATAG10-20000203/`.

[188] D. Van de Walle, N. Goeminne, F. Gielen, and R. Van de Walle. Challenges for Mobile Gaming based on AJAX, 2007. `http://www.w3.org/2007/06/mobile-ajax/papers/mobix.vandewalle.pdf`.

[189] P. Vuorimaa. Timesheets JavaScript Engine, 2007. `http://www.tml.tkk.fi/~pv/timesheets/`.

[190] P. Vuorimaa, D. Bulterman, and P. Cesar. SMIL Timesheets 1.0, 2007. `http://www.w3.org/TR/2008/WD-timesheets-20080110/`.

[191] K. Waters, R. Hosn, D. Raggett, S. Sathish, M. Womer, M. Froumentin, L. Rhys, and K. Rosenblatt. Delivery Context: Client Interfaces (DCCI) 1.0. W3C Working Draft, 2007. `http://www.w3.org/TR/2007/CR-DPF-20071221/`.

[192] The WebKit Open Source Project, 2007. `http://webkit.org`.

[193] C. Wei and J. Foncesa. Declarative Format for Applications and User Interfaces Use Cases and Requirements Version 1.0. *W3C Editor's Working Draft (document never published by the W3C)*, 2006. `http://dev.w3.org/2006/waf/DFAUI/DFAUI-UCs-and-Reqs.html`.

[194] C. Wenz. *Essential Silverlight*. O'Reilly, 2008.

[195] C. Wilson, P. Le Hegaret, and V. Apparao. Document Object Model (DOM) Level 2 Style Specification. W3C Recommendation, 2000. `http://www.w3.org/TR/2000/REC-DOM-Level-2-Style-20001113/`.

[196] D. Winer. OPML 2.0, 2007. `http://www.opml.org/spec2`.

[197] H. Wium Lie, T. Celik, and D. Glazman. Media Queries. W3C Candidate Recommendation, 2007. `http://www.w3.org/TR/2007/CR-css3-mediaqueries-20070606/`.

[198] Web Runtime (WRT) Desktop Development, 2007. `http://blogs.forum.nokia.com/blog/jari-otranens-forum-nokia-blog/browsing/`.

[199] XML Binding Language (XBL) 2.0. W3C Candidate Recommendation, 2007. `http://www.w3.org/TR/2007/CR-xbl-20070316/`.

All online references checked on April 15, 2008.

# Appendix A

# Markup and Code Examples

## A.1 CSS Media Types, Media Features and Implicit Transitions

In Listing A.1, media queries [197] are leveraged to serve `mobile.css` style sheet to user agents accepting media type `screen` or `handheld` and whose viewport width is 480 pixels or below. In Listing A.2, implicit transitions [103] are used to animate the enlargement of elements having a class `enlarge` while the pointer is hovered on top of them.

Listing **A.1** CSS media queries example.

```
<link media="handheld and (max-device-width: 480px),
            screen and (max-device-width: 480px)"
      href="mobile.css" type="text/css" rel="stylesheet" />
```

Listing **A.2** CSS implicit transitions.

```
.enlarge {
    width: 320px;
    height: 240px;
    -webkit-transition-property: all;
    -webkit-transition-duration: 2s;
    -webkit-transition-timing-function: ease-in-out;
}

.enlarge:hover  {
    width:  640px;
    height: 480px;
}
```

## A.2 S60 Web Runtime Widget Manifest

Widget properties supported in S60 Web Runtime 1.0 are listed in Table A.1 [52, p. 8]. An example of S60 Web Runtime `info.plist` manifest file is presented in Listing A.3.

Table A.1: Widget properties supported in S60 Web Runtime 1.0.

| Name | Type | Status | Example | Description |
|---|---|---|---|---|
| DisplayName | String | Required | Example Widget | Widget name, to be shown on the applications menu. |
| Identifier | String | Required | com.company.example | A unique string identifying the widget. |
| MainHTML | String | Required | example.html | Name of the HTML page to be loaded when the widget is initialized. |
| AllowNetwork Access | Boolean | Optional | true \| false | Defines is network access allowed. |
| Version | String | 1.0 | Optional | Widget's version. |

Listing **A.3** S60 Web Runtime manifest file example.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Nokia//DTD PLIST 1.0//EN"
    "http://www.nokia.com/NOKIA_COM_1/DTDs/plist-1.0.dtd">
<plist version="1.0">
<dict>
    <key>DisplayName</key>
    <string>Example Widget</string>
    <key>Identifier</key>
    <string>com.company.example</string>
    <key>MainHTML</key>
    <string>example.html</string>
    <key>Version</key>
    <string>1.0</string>
    <key>AllowNetworkAccess</key>
    <true/>
</dict>
</plist>
```

## A.3 Feed Widget Event Handling

Listing **A.4** A simplified example of the event handling mechanism of the Feed Widget.

```
<!-- index.html -->

<button id="doSomethingAction">Do Something</button>

/* MyController.js */

function MyController() {
var this.self = this;
}

MyController.prototype = {
    initialize: function () {
        var ed = new EventDispatcher(this.self);
        document.addEventListener('keydown',
                                  function(event) { ed.dispatch(event); },
                                  true);
    },

    doSomething: function () {
        // do something
    }
};

/* EventDispatcher.js */

function EventDispatcher(instance) {
    this.instance = instance;
};

EventDispatcher.prototype = {
    dispatch: function (event) {
        function execute(action) {
            instance[action](event.target);
        }
        function getActionName(element) {
         // returns the action name based on the id or className of the element
        }

        execute(getActionName(event.target));
    }
};
```

# Appendix B

# Rendering Engines

## B.1 Supported Web Technologies

Table B.1: Support for web technologies by rendering engines.

| Rendering engine | Trident | Gecko | WebKit | Presto |
|---|---|---|---|---|
| Browser version | Internet Explorer 7 | Firefox 2 | Web Browser for S60, Safari 2 | Opera 9 |
| XHTML | 1 partial | 1.1 | 1.1 | 1.1 |
| CSS | 2.1 partial | 2.1, 3 partial | 2.1, 3 partial | 2.1, 3 partial |
| JavaScript | 1.5 | 1.6 | 1.5 | 1.5 |
| DOM | 1, 2 partial | 1, 2, 3 partial | 1, 2, 3 partial | 1, 2, 3 partial |
| XForms | plugin | plugin | JS engine | JS engine |
| WICD | no | no | no | no |
| SVG | no | 1.1 partial | no | 1.1 partial |
| XSLT | yes | yes | no | yes |
| XPath | no | yes | no | yes |
| E4X | no | yes | no | no |