Web Application User Interface Technologies

Mikko Pohja

A" Aalto University

Web Application User Interface Technologies

Mikko Pohja

Doctoral dissertation for the degree of Doctor of Science in Technology to be presented with due permission of the School of Science for public examination and debate in Auditorium T2 at the Aalto University School of Science (Espoo, Finland) on the 4th of February 2011 at 12 noon.

Aalto University
School of Science
Department of Media Technology

Supervisor
Professor Petri Vuorimaa

Instructor
Professor Petri Vuorimaa

Preliminary examiners
Professor Tommi Mikkonen, Tampere University of Technology, Finland
D.Sc. Kari Systä, Nokia Research Center, Finland

Opponent
PhD, Research Director Fabio Paternò, Institute of the National Research Council of Italy

Abstract

The World Wide Web has expanded from a huge information storage repository into a worldwide application platform. Web applications have several benefits compared to desktop applications. An application can be used anywhere from any system and device, which means that only one version is needed, they do not need to be installed and developers can modify running applications. Despite all the benefits of the Web, web applications are suffering because they are developed using the same technologies as the static documents on the Web. Some of these web technologies are outdated and were not originally designed for the complex use cases of the modern applications to which they are now applied. For instance, HTML forms comprise the main interaction of an application, despite not having been designed to describe complex and interactive UIs. Another example is HTTP communication on the Web, which always requires client initiative and is too restrictive for dynamic web applications. Additionally, new usage contexts have brought with them new requirements for web applications, which are no longer used only via Graphical User Interfaces.

Recently, several parties have developed specialized technologies for web application development. These solutions are not only minor additions to the existing technologies, but also new technologies. The goal of this thesis is to analyze the advanced web technologies and propose improvements to the technologies and architecture where applicable. The technologies are evaluated against a large set of requirements. The aim of the evaluation is two-fold. The first part is to select a technology on which to base the further improvements, and the second is to identify the deficiencies of the current solutions. The improvements focus on the developers' point-of-view.

Based on the evaluation, this thesis proposes certain improvements related to multimodal interaction, server push, and remote UI updates. It also discusses software that supports the improvements and XML-based web technologies. Finally, the improvements are evaluated against the requirements and compared to other solutions.

Tiivistelmä

Web ei ole enää pelkkä suuri tietovarasto kuten aikaisemmin, vaan myös maailmanlaajuinen sovellusalusta. Web-sovelluksilla on useita etuja työpöytäsovelluksiin nähden. Sovelluksen käyttäminen ei ole sidottu tiettyyn paikkaan, järjestelmään tai laitteeseen. Käyttäjän ei tarvitse erikseen asentaa sovelluksia ja kehittäjä voi muokata käytössä olevaa versiota sovelluksesta. Eduista huolimatta sovelluskehitystä haittaavat siinä käytettävät teknologiat. Sovelluksia kehitetään pääasiassa samoilla tekniikoilla kuin staattisia web-sivuja. Osa kyseisistä tekniikoista on vanhentunut, eikä niitä ole edes suunniteltu käytettävän sovelluksissa. Esimerkiksi sovellusten vuorovaikutus toteutetaan HTML-lomakkeilla, vaikkei niitä ole suunniteltu kuvaamaan monimutkaisia ja vuorovaikutteisia käyttöliittymiä. Toinen esimerkki on webin HTTP-pohjainen tiedonvälitys, joka perustuu asiakkaan tiedonvälityspyyntöihin palvelimelta. Tällainen tiedonvälitysmalli rajoittaa liikaa dynaamisia web-sovelluksia. Myös uudet vuorovaikutustavat asettavat tekniikoille lisävaatimuksia. Sovelluksia ei käytetä enää pelkästään graafisen käyttöliittymän avulla.

Web-sovellusten kehittämiseen on viime aikoina määritelty useampia ratkaisuja. Ne eivät ole pelkästään parannuksia olemassa oleviin tekniikoihin, vaan myös kokonaan uusia tekniikoita. Tämän työn tarkoituksena on analysoida näitä edistyneempiä tekniikoita ja ehdottaa niihin tarvittaessa parannuksia. Arviointiin käytetään erikseen määriteltyjä vaatimuksia. Arvioinnilla on kaksi tarkoitusta: yhtäältä, valita teknologia, jota tässä työssä käytetään hyväksi, ja toisaalta, löytää nykyisten ratkaisujen puutteet. Parannuksia mietitään lähinnä sovelluskehittäjän kannalta.

Arviointeihin perustuen tässä työssä esitetään parannuksia monimuotoiseen vuorovaikutukseen, palvelinlähtöiseen tiedonvälitykseen ja käyttöliittymän etäpäivittämiseen. Lisäksi kuvataan ehdotukset toteuttavat ohjelmistot. Parannusehdotukset myös analysoidaan vertaamalla niitä vaatimuksiin ja muihin ratkaisuihin.

Acknowledgements

I would like to express my gratitude to Professor Petri Vuorimaa, who has supervised this Thesis. He gave me opportunity to work as a researcher and has supported and guided me during the work. Without him, this Thesis would never have been materialized. Furthermore, I would like to acknowledge the research group, where I started this work. As a member of the group, I learned how to do research and write scientific articles. Especially, senior researchers Mikko Honkala, who has co-authored two articles of this Thesis, Pablo Cesar, and Kari Pihkala have familiarized me with the research work. Moreover, I worked closely with Alessandro Cogliati, Teppo Jalava, and Juha Vierinen.

The pre-examiners, Professor Tommi Mikkonen and D.Sc. Kari Systä, affected significantly to the final Thesis. They gave me good feedback and constructive comments, for which I am very thankful. I got valuable comments from the final revisions of the Thesis also from my colleagues Denis Shestakov and Kalle Säilä, who I also would like to thank.

Espoo, January 17, 2011

Mikko Pohja

Contents

List of Publications

This thesis consists of an overview and of the following publications which are referred to in the text by their Roman numerals.

I Mikko Pohja, Mikko Honkala, and Petri Vuorimaa. An XHTML 2.0 Implementation. In *Proceedings of the Fourth International Conference on Web Engineering (ICWE)*, pages 402-415, 2004.

II Mikko Pohja and Petri Vuorimaa. CSS Layout Engine for Compound Documents. In *Proceedings of the Third Latin American Web Congress (LA-WEB)*, pages 148-157, 2005.

III Mikko Honkala and Mikko Pohja. Multimodal Interaction with XForms. In *Proceedings of the Sixth International Conference on Web Engineering (ICWE)*, pages 201-208, 2006.

IV Mikko Pohja. Declarative Push on Web. In *Proceedings of the Fourth International Conference on Web Information Systems and Technologies (WEBIST)*, 201-207 2008.

V Mikko Pohja. Comparison of Common XML-Based Web User Interface Languages. *Journal of Web Engineering*, Vol. 9, No. 2, 2010, pages 95-115, 2010.

VI Mikko Pohja. Server Push for Web Applications via Instant Messaging. *Journal of Web Engineering*, Vol. 9, No. 3, 2010, pages 227-242, 2010.

Summary of the Publications and Author's Contribution

Publication I: "An XHTML 2.0 Implementation"

The paper discusses the XHTML 2.0 and XForms specifications and their user agent implementation. It also evaluates the impacts of transition to XHTML 2.0 both authors' and browser manufactures' points of view. The author has designed and implemented the XHTML component and integration of the XHTML and CSS layout. He has written half of the article.

Publication II: "CSS Layout Engine for Compound Documents"

Specific purpose XML languages can be combined as needed. The resulting documents are called compound documents. The paper defines requirements for the XML compound document's layout engine. The requirements relate to the features of the layout engine, devices where it can operate, and the interfaces of the cooperating components. The paper also describes an implementation of the layout engine, which conforms to the requirements. The author has designed and implemented the CSS layout engine described in the paper. In addition, he has written all of the text, while getting comments and proofreading help from Prof. Petri Vuorimaa.

Publication III: "Multimodal Interaction with XForms"

Introduces a model and an implementation of XFormsMM, which includes XForms 1.0 combined with modality-dependent stylesheets and a multimodal interaction manager. The model can be used to create multimodal applications using a write once approach. The model is evaluated com-

paring it with SALT and X+V multimodal authoring models based on the W3C's Multimodal Interaction Requirements and implementing two use cases. The author has assisted to design and co-developed the described multimodal framework with Mikko Honkala.

Publication IV: "Declarative Push on Web"

Defines and discusses four methods of using a declarative description of push-updates for Web documents. The methods are defined by combining existing and upcoming web technologies. The scope of the paper is on targeting the update and on modifying the document on the client side. To evaluate the methods, a use case is designed and implemented with all the methods. The author has been a sole author of the article.

Publication V: "Comparison of Common XML-Based Web User Interface Languages"

Evaluates five XML-based UI description formats, HTML5, XForms, XAML, LZX, and XUL, in order to determine which language is best suited for modern web application development. The paper also assesses what kind of applications are suited to each format. The requirements for a Web UI description language from the literature are revised and three use cases are defined, through which the languages are evaluated. The article is an extension to a prior conference article [71], which was comprised of evaluation of two UI formats. The extended article adds three more formats to the comparison. The author has been a sole author of the extended article.

Publication VI: "Server Push for Web Applications via Instant Messaging"

The paper evaluates how an instant messaging protocol, XMPP, can complement HTTP-based web applications. Through XMPP, a server-side component of a web application is able to push information to the client side. The paper presents a communication paradigm of a push system and an implementation of it. In addition, another communication paradigm is

sketched for inter-widget messaging on the Web. Based on that paradigm a new research problem is defined and presented. The author has been a sole author of the article.

List of Abbreviations

2D	Two dimensional
API	Application Programming Interface
BOSH	Bidirectional-streams Over Synchronous HTTP
CSS	Cascading Style Sheets
DBMS	Database Management Systems
DHTML	Dynamic Hyper Text Markup Language
DOM	Document Object Model
GUI	Graphical User Interface
HTML	Hyper Text Markup Language
HTML5	Hyper Text Markup Language revision 5
HTTP	Hypertext Transfer Protocol
ID	Identification
IM	Instant Messaging
LZX	Laszlo XML
MIME	Multipurpose Internet Mail Extensions
MLFC	Markup Language Functional Component
PDA	Personal Digital Assistant
PHP	PHP: Hypertext Preprocessor
Pub/Sub	Publish/Subscribe
RDE	Remote DOM Events
REST	Representational State Transfer
REX	Remote Events for XML
SALT	Speech Application Language Tags
SGML	Standard Generalized Markup Language
SMIL	Synchronized Multimedia Integration Language
SQL	Structured Query Language

SSE	Server Sent Events
SVG	Scalable Vector Graphics
UI	User Interface
UIDL	User Interface Description Language
URI	Universal Resource Identifier
W3C	World Wide Web Consortium
WPF	Windows Presentation Foundation
WWW	World Wide Web
X+V	XHTML+Voice
XAML	Extensible Application Markup Language
XBL	Extensible Binding Language
XHTML	Extensible Hyper Text Markup Language
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol
XPCOM	Cross Platform Component Object Model
XPConnect	Cross Platform Connect
XPInstall	Cross Platform Install
XUL	XML User Interface Language

1 Introduction

As well as being a storage site for information, the World Wide Web (WWW or Web) is increasingly becoming used as an application platform. Commerce and communication tasks, such as the use of e-mail, have become common on the Web, as have tasks with higher interaction, such as information authoring [32]. The Web has expanded, therefore, from a platform for information storage into a platform for distributed applications.

Migrating applications to the Web can have many benefits; for example, the same application can be used anywhere from any system and device [48], which substantially reduces development costs compared to the development of desktop applications. Because web applications are run from servers, there is no need to install them; a user agent can access the applications directly with only a web browser. This means that web applications are easy to try, for instance, before committing to purchase them. Furthermore, it is easy to add new features and fix bugs because the developer only has to modify a single instance of an application in order for the application to run effectively [48]. As well, software updates are automatic; that is, users and administrators do not have to do anything in order to upgrade an application.

The key requirement for multiuser applications is that they share the same data [67]. As they reside on servers, web applications are easy to realize as multiuser applications. The data is located on a server and is transferred to the clients for operations. It is usually safer to retain the data on a server than to keep it on the client side, as is the case with desktop applications. This is due to the fact that modern servers and their hard drives are carefully backed up. Moreover, security is usually better in professionally maintained computers, which makes the applications less prone to viruses. Finally, the fact that all the users are using same version of the application usually means that there are less errors in the software.

1.1 Motivation

Even though web applications have many advantages over desktop applications, the underlying technologies and architecture continue to hinder full exploitation of the possibilities of the Web. Some web technologies are outdated and were not originally designed for the complex use cases of modern applications. Hyper Text Markup Language (HTML) [75] forms, for example, which comprise the main interaction of an application, were not designed to describe complex, higher-interaction UIs. The use of HTML forms, along with client-side scripting, has led to poor usability, maintainability, re-use and accessibility [81]. Another example is the Hypertext Transfer Protocol (HTTP) [25] -based communication on the Web. By design, clients must request data in HTTP. However, there are a lot of use cases in which a server should be able to send data immediately as a reaction to events, without having to wait for the client's request [37].

A growing number of devices, including mobile phones and Personal Digital Assistants (PDA), have access to the Web. This has made web applications even more widespread and it also means that web applications must run on a variety of platforms. When designing a web application, therefore, one must consider the varying capabilities of certain devices, such as display size, local storage size, method of input, and network capacity [26].

New usage contexts also raise new requirements for web applications [31]. Using multiple modalities (such as aural and visual input and output) simultaneously can help fulfill these requirements. For instance, a keyboard-less device could use voice input to enhance the input capabilities of a graphical application. Studies have also shown that new and complex tasks are solved faster by using multiple modalities [68]. The problem is that current approaches to authoring multimodal Web applications concentrate on one modality at a time. For example, the HTML+ECMAScript approach focuses on the visual User Interface (UI) and has well-known accessibility issues when used with speech tools.

Several parties have developed technologies to overcome the problems that have been noted above. Rather than minor additions to the existing technologies, these solutions are completely new technologies and are referred to in this Thesis as *advanced web technologies*. Due to the presence of several candidates for solving the problems, this Thesis does not attempt to propose yet another technology. Instead, the goal of this Thesis

is to analyze the existing advanced web technologies and propose improvements to the technologies and architecture where applicable.

1.2 Background

The Internet, which was formed by interconnected computer networks, provides a massive worldwide platform for communication between computers and, therefore, a global platform for distributed applications. The Web is the application that lies on top of the Internet. Initially, the Web was a collection of static documents that were virtually connected to each other via hyperlinks, but now, it is used more and more as a worldwide application platform [32]. This Section reviews the building blocks of the Web and web applications.

1.2.1 Foundations of the Web

Even though the Web has expanded from an information storage repository to an application platform, it continues to rely on its fundamental concepts: Uniform Resource Identifier (URI) [7], HTML, and HTTP. These three key technologies still provide the means for identifying, transferring and describing documents, or applications for that matter, on the Web [46]. Figure 1.1 depicts the technology stack of the Web.

Presentation	(X)HTML	CSS
Logic		ECMAScript
Structure	XML, DOM
Identification	URI
Communication	HTTP
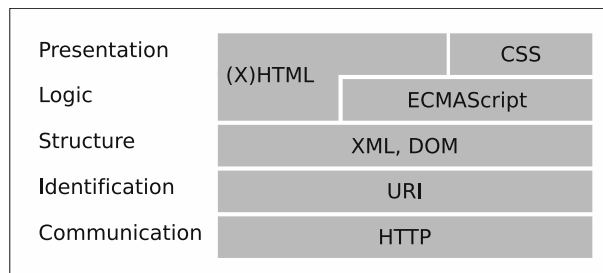
Figure 1.1: Technology Stack of the Web.

Web agents deliver resources through HTTP and resources are identified using URI. The resources can be in any format. There are certain advantages to using formats based on Extensible Markup Language (XML) [13] because, almost without exception, the user agents provide Document Object Model (DOM) [3] processing. However, it is better to present some media, such as images and video, in binary form. The logic

of the applications is typically defined by ECMAScript [22] on the client side, while there are a number of options to choose from on the server side. ECMAScript on the client side controls the presentation layer, which is described by an HTML document and Cascading Style Sheets (CSS) [10] style declarations [46].

1.2.2 Architecture of Web Applications

Web applications generally adopt a traditional three-tier architecture [68] that distinguishes between presentation logic, business logic, and data logic. The presentation layer contains UI and controls user interaction, the logic tier contains business logic and the data tier provides data access. The three separate logic layers use abstract interfaces to communicate with each other, which makes it easy to change the implementation of one layer without affecting the others.

Figure 1.2 depicts an adoption of the three-tier architecture to the Web environment. The presentation layer consists of a user agent, while the logic tier is comprised of a web server and an application server, which can be separate components. In addition, due to the architecture of the Web, the logic tier is also partitioned more and more by HTTP to the client side [100]. This is a consequence of the fact that it is beneficial to define certain application logic on client side because computing on server side always requires a request over a network. The amount of client-side computing varies greatly between applications (see below for further discussion of this subject). Because the data tier, which stores the application data on the server side, is not affected by web technologies or the architecture of the Web, generic databases are usually used.

Server-side Model

In the classical server side model, static HTML and CSS documents describe the UI and application logic is entirely on the server side [76]. The interface is clear and a developer can select from various technologies with which to develop his or her application. The downside is that user actions are also handled on the server side, which naturally causes considerable latency for user interaction.
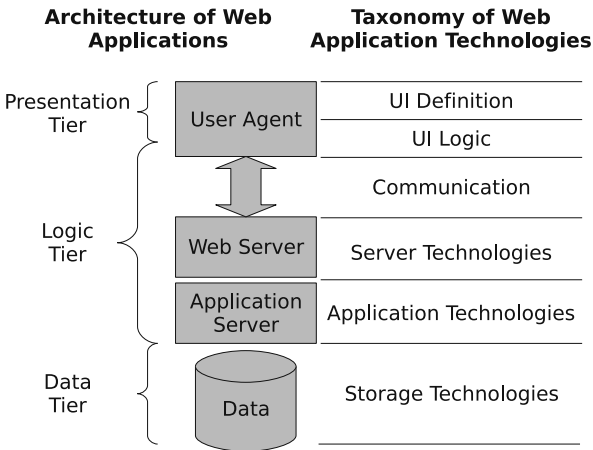
Figure 1.2: Architecture of web applications and taxonomy of web application technologies.

Ajax Model

An Ajax model tackles the latency problem of the server-side model. It adds an intermediate component on the client side, which can asynchronously communicate with the server-side process. Adding a functional component to the client side naturally requires more processing power from the client.

Client-Side Model

Mikkonen and Taivalsaari have proposed that all the functionality should reside on the client side except for data validation and storing [59]. The benefits of this method include the fact that it is easier to develop software using a single paradigm rather than dividing it into multiple languages and sites. Secondly, it reduces the communication with the server since client-side logic can respond to the user actions. Furthermore, it is even possible to use the applications offline to some degree. The client-side model is emerging along with more powerful and sophisticated user agents, the so-called thick clients. The different approaches are discussed below.

The Lively Kernel [44] is a platform for web programming written in Javascript. It not only makes it possible to run web applications written for it, but also to develop the applications on the platform. Because it is based on Javascript, there is no need to install or compile the applications at all. This proves that a web browser can act as an application platform

alone.

There are also commercial platforms that are aimed at the thick clients, including Microsoft Silverlight, XULRunner, Sun JavaFX, and Adobe AIR. They extend browsers in such a way that they can also contain the application logic. Typically, proprietary languages define either application logic or UI or even both.

There are Javascript libraries like JQuery [89] or Prototype [72], which assist in web application development. They help with implementation, such as communication and UI components and actions as well as taking care of browser incompatibilities instead of a developer.

The XFormsDB [42] framework provides a means with which to describe web applications declaratively on the client side and it extends the XForms 1.1 specification [11]. The extensions include access to a data source, authentication, access control, state handling, and session management. XFormsDB provides a single paradigm with which to describe multiuser Web applications like the Lively Kernel, albeit declaratively. This makes it possible to create an editor with which to graphically edit the XFormsDB applications.

Synchronized Multimedia Integration Language (SMIL) 3.0 specification [14] contains a SMIL State module that can be used along with the other specifications to create adaptive time-based web applications [47]. The article shows a feasibility of declarative programming when the scope of language is well defined. It might be possible to use the same functionality with any XML language via SMIL Timesheets [94].

1.2.3   Web Application Technologies

Creating a web application requires a large number of technologies. In Figure 1.2, the web application technologies have been divided into six categories, each of which contains many technologies and most have several options from which to choose. The taxonomy includes two client-side categories (UI definition and UI Logic), a middle category that contains technologies for communication between client and server, and three server-side categories, namely web servers, applications and storage technologies. The categories and the corresponding technologies are reviewed below. This Thesis uses the following terminology:

Web Application Technology: A technology from any category of the

taxonomy shown in Figure 1.2.

Web Technology: A technology used on the presentation tier of the web applications (cf. Figure 1.2). In the taxonomy, the web technologies belong either to UI definition or UI logic class.

Advanced Web Technology: A novel *Web Technology* which has been developed especially for web application development and to overcome the problems of the legacy web technologies.

The technologies in the scope of this Thesis, namely web technologies and communication technologies, are discussed in more detail in the next Chapter.

Web Technologies

The UI of a web application consists of different UI elements, their presentation, and interaction. HTML is currently used with CSS and ECMAScript to create web applications. This technology 'troika' is usually referred to Dynamic HTML (DHTML) [96] and it partially fulfills the World Wide Web Consortium's (W3C) requirement to separate content, presentation, and interaction [46]. Even though HTML mainly describes the content, it also has some presentational aspects.

Communication

Web applications use HTTP protocol to communicate between the presentation and the logic tier. Generic software handles the communication, which means that an author does not have to implement it. HTTP is an application-level protocol for distributed, collaborative, hypermedia information systems used for data transfer in the Web.

Web Server

A web server is a counterpart that allows a user agent to realize communication over a network. A web application author does not typically have to implement anything for the web server, instead they just configure it to identify the resources of the application. Basically, web servers include a standard HTTP implementation, which can handle requests from clients and send responses back to them. Additionally, the web servers usually provide interfaces to support dynamic content in addition to static content, authentication, and secured connections. A number of technologies exist for the dynamic creation of content on the server side; these are

discussed in the next section of this thesis. Finally, web servers may support virtual hosting, content compression and other things that may help manage client-server communication.

Application Server

The business logic of an application is traditionally defined on an application server. There are a number of technologies with which a developer can implement the logic, including PHP, Java, C#, Perl, Ruby, and Cold-Fusion. The technologies are often a part of a web application framework, which is a reusable and modular platform that can be specialized to produce custom web applications [84]. Frameworks help ease the web application development considerably and make it possible to generate code for other layers of an application [48].

Storage Technologies

Web applications typically use regular Database Management Systems (DBMS) to store application data. DBMS often provide database server functionality, which means that they rely on the client-server model for database access.

Recently, a growing number of web applications have had persistent storage on the client side for storing user-related data, for instance for offline use. W3C has started to standardize the technologies. Web Storage [38] specification defines an API for storing key-value pair data in web clients, while Indexed Database API [56] and Web Structured Query Language (SQL) Database [39] specifications define more versatile data querying options. Indexed Database API provides indexed records of data and a query language can be layered on top of the API. Web SQL Database uses SQLite [18] for queries.

1.3 Research Problem and Aim

As discussed above, most of the current web technologies were not originally designed for the complex use cases of today's web applications. For example, they do not lend themselves particularly well to the creation of higher-interaction and multimodal UIs, while the communication methods are not sufficiently versatile. Although, in theory, the Web provides powerful platform for applications, in practice the web technologies have

certain deficiencies that remain to be solved. Accordingly, the main research question of this Thesis is:

How can web user interface technologies be improved in order to respond to the demands of today's web applications?

To answer this question, the following sub-questions are considered:

Q1: What is required from the web user interface technologies to support today's web applications?

Q2: What deficiencies exist in the web user interface technologies?

Q3: How can the identified flaws be fixed?

Q4: How can the proposed improvements be implemented?

In short, this Thesis aims to analyze advanced web technologies and find solutions to the limitations of these technologies.

1.4 Research Methods

This Thesis is the result of evaluation and concept formulation research approaches [93]. The results of the evaluation approaches are utilized by concept formulations. The main research methods are conceptual analysis and concept implementation.

The research work was divided into a number of phases, which resulted in several publications. Each phase contained a background of a topic reviewed via the literature and, typically, the validation was based on requirements obtained from the literature review. Alternatively, the requirements were defined or complemented during the author's work. Depending on the phase, pertinent requirements were used either to evaluate existing technologies or to define new ones. In addition to the requirements, the work items were validated through proof-of-concept implementations, which provide information about the implementation issues of the proposals.

1.5 Scope of the Research

This Thesis focuses on web technologies; that is, how a user interface and a required client-side logic of a web application can be defined. Furthermore, when applicable, it also addresses user agents' communication with a back-end system. All of these technologies belong to the three highest categories of the taxonomy of web application technologies depicted in Figure 1.2. The focus of the Thesis is on improvements to advanced web technologies. In other words, the Thesis presupposes that some legacy web technologies will give way to new and advanced web technologies, as defined by several parties. Given the number of advanced web technologies, however, the Thesis seeks to improve the existing technologies rather than attempt to define yet another technology.

This Thesis does not contain an extensive survey of web application UI technologies; instead it offers a comparison of common XML-based technologies. The rationale behind this decision is to focus on technologies that have widely available implementations. The aim is also to compare technologies that represent different development models. Finally, the Thesis do not attempt to evaluate what would be required to adopt a new technology on a large scale, but focuses on technical merits of the technologies.

The Thesis mainly takes a developer perspective because developers mostly come into contact with the specific features of technologies. Although functionality can typically be implemented with any technology, the amount and difficulty of coding and, later, also maintenance may vary greatly. In other words, the end-user may not even recognize the technology with which an application has been implemented, although it may have a huge impact for a developer.

1.6 Contribution

This Thesis contributes on several levels. Firstly, it collects a wide set of requirements regarding web application user interface technologies. These requirements are gathered from the literature and revisited by the author and are used to evaluate existing technologies, define new ones or both. Advanced web application user interface languages are evaluated thoroughly, based on which improvements are proposed to the advanced

web application user interface technologies. The Thesis introduces these proposals at a generic level and shows their proof-of-concept implementations. The Thesis ends by validating the proposals. The main contributions and the relating publications are as follows:

Requirements for web application UI technologies. The requirements cover web UI languages, multimodal technologies, and server push systems. Publication V collects the web UI language requirements from the literature and completes them. The requirements for the server push systems are defined in Publication VI, whereas multimodal technology requirements are solely from the literature.

Evaluation of the advanced web application UI technologies. The requirements above are used to evaluate the web UI languages, their communication with the back-end, and specific multimodal web technologies. The first two are originally evaluated in Publication V and the last one in Publication III.

Proposed improvements to the web UI technologies. The improvements relate to multimodal interaction, declaratively defined UI updates, and a server push system. Publication III defines a multimodal framework based on XForms language. The framework can be used to create multimodal applications using a write once approach. Publication IV discuss how declaratively defined UI updates should be handled on the client side. The proposed server push system has been defined in Publication VI.

Implementations of the proposed improvements. The proposed improvements have been implemented as proofs of concepts. The multimodal framework is implemented on top of an XML browser. The browser's XHTML+XForms compound has been discussed in Publication I and the compound's layout engine in Publication II. The implementation of the framework has been described in Publication III. The support for declaratively defined UI updates have been discussed in Publication IV and Publication VI represented a prototype implementation of the server push system.

Evaluation of the improvements. The improvements and their implementations have been evaluated at the end of the Thesis. Each publication above representing an implementation also evaluates the respective proposal and the implementation.

1.7 Organization of the Thesis

The rest of the Thesis is organized as follows. Next Chapter reviews the state of the art of the web technologies. Chapter 3 collects the requirements for web technologies to answer the Research Question Q1 and evaluates a set of advanced web technologies (Q2). The Research Question Q3 is addressed in Chapter 4, which proposes improvements for the advanced web technologies. Implementations of the proposed improvements (Q4) are discussed in Chapter 5 and the results are evaluated in Chapter 6. Author's contribution and future work are given in Chapter 7.

2 State of the Art

This Chapter reviews the state of the art of the technologies in the scope of the Thesis. That includes UI and communication technologies of the web applications. The Chapter also summarizes the problems of the current solutions.

2.1 User Interface

The UIs of web applications are defined in three technologies, namely HTML, CSS, and ECMAScript. The purpose of the technologies are content definition, layout definition, and UI logic, respectively. This Section discusses also programming paradigms of the web UI development. Currently, imperative and declarative paradigms are mixed in the web UI development that makes the development more complicated.

2.1.1 HTML

HTML is a document description language. HTML elements describe document objects like headings, paragraphs, images, etc. It also contains notation for hyper links from which the name of the language has been originated. The main user interaction method of the HTML is forms.

When the Web was still in its infancy, HTML was used to handle all the aspects of the web pages or documents [74]. At that time, the Web was comprised of static documents, so a language that described content was perfectly sufficient. Even though the Web has also become an application platform, it still heavily relies on this document description language. Naturally, HTML has experienced considerable development during the evolution of the Web and the definition of separate technologies for presentation (CSS) and interaction (ECMAScript) has lightened its burden. Still, its original purpose was to describe documents, not application UIs.

It lacks several elements that are popular in today's UIs and its user interaction capabilities are limited. The main reason why it remains the first choice for UI description on the Web is the desire to be backward-compatible with the legacy content on the Web.

HTML 4.01 is the latest HTML version and Extensible Hyper Text Markup Language (XHTML) is its XML-based counterpart. There are two versions of XHTML. Version 1.0 [69] defined HTML according to the XML syntax and XHTML 1.1 [1] modularized the language. At the time of writing, a new version of HTML, HTML5, is a work in progress at W3C. The aim is to fix errors and problems with the previous version and add new features especially for web applications. HTML5 is one of the formats evaluated in this Thesis and it is reviewed on more detail in the next Chapter.

2.1.2 CSS

CSS [10] is a mechanism for styling web documents. CSS enables the style of a web document to be separated from the content. This separation aims to improve content accessibility, provide greater flexibility and control in the specification of presentation characteristics, enable multiple pages to share formatting using intelligent selectors, and reduce complexity and repetition in structural content. Cascading the styling rules means that the styles can be brought in from several sources and they are cascaded in a user agent as one. CSS also allows the same markup page to be presented in different styles for different rendering methods, such as on-screen, in print, by voice, and on Braille-based tactile devices. These options make site maintenance easier and web authoring simpler.

At the moment, the most common browsers have reasonably complete levels of implementation of CSS level 2 revision 1 [10]. The specification is at Candidate Recommendation stage at W3C and it will reach a Recommendation stage when there are two complete independent implementations of it. In addition, CSS level 3 is also a work in progress at W3C. It broadens the scope of the CSS. One of the goals is to reduce scripting for styling and use native CSS implementation instead. The CSS level 3 specification has been divided into several modules, each of them developed as a separate specification. The major browsers have partial implementations of the different CSS level 3 modules.

2.1.3 ECMAScript

ECMAScript is a prototype-based object-oriented scripting language used to implement web application logic on the client side. ECMAScript is interpreted in runtime, unlike many popular programming languages like C or Java that must be compiled prior to execution. Runtime interpretation allows the code to be modified, for example, by adding functions and variables, even in runtime. The two best-known implementations of ECMAScript are Microsoft's JScript [57] and Netscape's JavaScript [63], on which ECMAScript was originally based.

ECMAScript is constantly developed further. The aim is to make it a generic programming language, not just a DOM processing tool [23]. The current version of the ECMAScript, 5th Edition [22], is partially supported by major browsers. The next version, code named 'Harmony', is a work in progress at ECMA International.

2.1.4 Imperative versus Declarative Approaches

In addition to features, technologies differ on a more fundamental level, namely the kind of model an author uses to describe the UI. Traditionally, in desktop programming, the UI has been just another program component that has been implemented using a UI toolkit with the same language as the components. That is an example of imperative programming. The same can also be applied on the Web using ECMAScript. The entire DOM, which represents the UI on runtime, can be formed with ECMAScript instructions.

The other model used to program UIs is a declarative model. The main difference between imperative and declarative approaches is the control of the program [4]. In declarative languages, a programmer only provides the logic of the program and the control is left to the language. Leaving the control to the language means that a predefined set of functions can be used. With imperative languages, the programmer must also implement the control. Although this means that they are more powerful, they are also harder to learn [51] and maintain, and are error-prone. It has been said that people who cannot do programming can still utilize declarative languages. That is due to the aforementioned fact that, with declarative languages, a user defines only what happens, not how does it happen. Furthermore, there is a limited set of options what can be defined

to happen with a declarative language.

HTML is a declarative language. The presence of billions of web pages has proved that basically anyone can author HTML and publish it on the Web. However, that applies mainly to static documents. More application-orientated DHTML also contains ECMAScript part, which is not declarative. This can prevent non-programmers from creating applications on the Web.

Schmitz has listed reasons why the declarative approach is better than the imperative approach in UI development [82]:

1. UI developers are not programmers,

2. Authoring tool support requires a declarative solution,

3. Declarative solutions will speed up adoption and deployment in the marketplace,

4. UI performance through a professionally developed user agent is more robust and predictable, and

5. Declarative approach is more secure, since the syntax is bounded.

2.1.5 Web Application Programming Interfaces

The DOM [3] is a runtime object model of content and structure of the web documents. User agents parse HTML and XML into a DOM, whereby the documents can be modified in runtime. A sample HTML document is shown in Listing 2.1 and a corresponding DOM is depicted in Figure 2.1. The text of the document has been structured by HTML tags in Listing 2.1. As can be seen from the Figure, the HTML tags are represented as elements in the DOM tree and text snippets of the document form the text nodes of the DOM.

Listing 2.1: Sample HTML markup.

```
1  <html>
2    <body>
3      <h1>Lorem</h1>
4      <p>Ipsum <i>dolor</i> sit.</p>
5    </body>
6  <html>
```

A user agent renders a document according to its DOM. Afterwards, the layout and user interaction among the others can be controlled via specific DOM Application Programming Interfaces (API). The DOM APIs are typically utilized by ECMAScript functions on the client side. W3C has defined several language neutral APIs with which to access the object model. The APIs address general HTML and XML processing, HTML specific APIs, and event model structured languages.
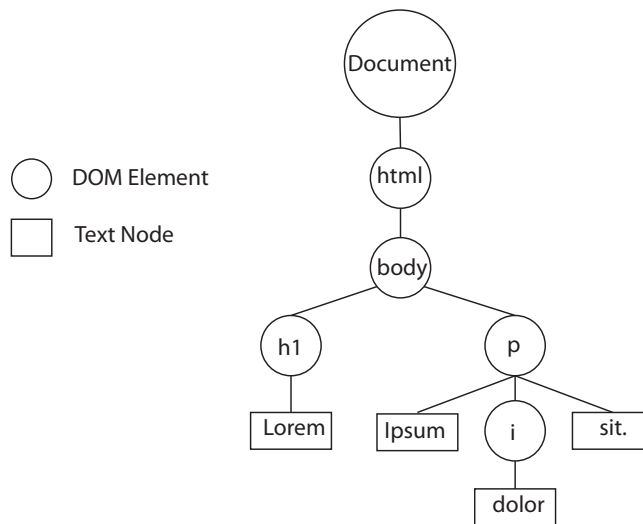

Figure 2.1: DOM tree of the document in Listing 2.1.

2.2 Communication

Communication on the Web is based on the request-response paradigm [25]. That is, a client asks for information from a server and the server submits the information to the client as a response. Although this method is sufficient for downloading static web documents, web applica-

17

tions require more versatile communication. In addition to downloading a web application, there might be a need for a client to request incremental updates or for the server to automatically submit updates to the application on the client side. For example, a chat application must push all the messages in real time to all participants.

2.2.1   HTTP

HTTP protocol is a request-response protocol. A client sends a request to the server in the form of a request method, URI, and protocol version, followed by a Multipurpose Internet Mail Extensions (MIME) [29]-like message. The server responds with a status line, including the message's protocol version and a success or error code, followed by a MIME-like message. Usually, a user agent initiates the HTTP communication. [25]

   Problems arise when HTTP is used for an application that is distributed between the client and the server. For example, user operation on the client side may require an action on the server side. To initiate the action, the client must send another HTTP request to the server and wait for the response before the operation can be completed. This also means that the entire document has to be downloaded and rendered again, which is time-consuming and can reduce the responsiveness of the user interface. The performance can be improved by updating only a part of the user interface using the Ajax framework [30].

2.2.2   Asynchronous Communication

A common problem with distributed applications is a latency caused by communication of separated components. Since communication often leads to a user action, user experience is decreased if the user has to always wait for a response to the actions. A web application can communicate with a server asynchronously with XMLHttpRequest [92]. Together with DOM, XML, and ECMAScript they form a framework, for which Garrett coined the term 'Ajax' [30] in 2005. An Ajax engine on the client side can communicate with the server without interrupting the use of an application. This means that all user actions that require server interaction can be realized with the Ajax.

2.3 Summary

This chapter has reviewed the state of the art of the web technologies. The aim of this Thesis is to improve technologies in the UI definition, UI logic, and communication categories and, to this end, Table 2.1 presents the main problems in the categories within the scope of the thesis are listed in.

Table 2.1: Summary of problems identified in current technologies within the scope of this Thesis.

Category	Problem	Description
UI Definition	Separation of Content and Presentation	HTML mixes content and presentation.
	HTML Forms	Not designed to describe complex, high-interaction UIs.
	Multimodal Interaction	Current web UI technologies support only graphical modality.
	Purpose of HTML	HTML is for documents not web applications.
UI Logic	Syntax of the logic	Imperative languages are difficult to maintain and they are error-prone. Non-programmers cannot utilize them.
Communication	Server Push	Server push must be emulated by pull.

3 Evaluation of the Advanced Web Technologies

Several technologies have been developed in order to overcome the problems presented in the previous Chapter. The current Chapter discusses those technologies, referred as *advanced web technologies*. Furthermore, the Chapter also defines requirements against with the technologies are evaluated.

3.1 User Interface Languages

Declarative user interface descriptions predate the advent of the WWW and HTML. They are used as part of a model-based UI design, in which the UI is built on a certain model. Among other things, the model can be a description of tasks, data or a presentation. The aim is to identify reusable components of the UI and to capture more knowledge in the model [27]. The author can specify what features the interface should have, rather than write programs that define how the features should work, which saves them from writing a lot of procedural code [73]. One study showed that an average of 48 percent of an application's code is devoted to the user interface portion [64]. Enhancing UI development has the potential to boost the application development process considerably.

W3C hosted a Model-based User Interfaces incubator group in order to determine whether there should be a specification of model-based UIs for web applications. On of the group's conclusions was that there are several items in the context of Model-based UIs that should be standardized [28]. In addition, the ACM Transactions on Computer-Human Interaction journal recently published a special issue on User Interface Description Languages (UIDL) for Next Generation User Interfaces [83] that introduces novel UIDL approaches. This Thesis studies common XML-based languages that have readily available cross-platform implementations, a criterion that rules out some research-oriented UI languages. Popular Web

toolkits such as Dojo [20] and Google Web Toolkit [34] are not included because their outcome is usually HTML + Javascript, which is naturally close to HTML5 [40]. The selected approaches are introduced below.

3.1.1 XForms

XForms 1.0 Recommendation [21] is the next-generation Web forms language, designed by the W3C. It solves some of the problems found in the HTML forms by separating the purpose from the presentation and using declarative mark-up to describe the most common operations in form-based applications [15]. It can use any XML grammar to describe the content of the form (the instance data), which also enables the creation of generic editors for different XML grammars with XForms. It is possible to create complex forms with XForms using declarative mark-up without resorting to scripting.

XForms is an abstract user interface description language, one of the design goals of which was to avoid mandating a certain modality. This means that it can be suited to describing user interfaces, which are realized in different modalities, such as the Graphical User Interface (GUI) and Speech.

Several XML vocabularies have been specified in W3C. Typically, an XML language is targeted for a certain purpose (e.g., XHTML for content structuring or Scalable Vector Graphics (SVG) for two dimensional (2D) graphics). XML languages can also be combined. A compound document, which is an XML document that consists of two or more XML languages, can specify the user interface of an application. In this Thesis, XForms is combined with XHTML+CSS level 2. XForms 1.0 directly includes the following W3C specifications: XML Events [54], XPath 1.0 [16], XML Schema Datatypes [8], and XML 1.0.

3.1.2 XUL

Mozilla has developed a UI description language called XML User Interface Language (XUL) [33]. The mark-up consists of widget elements such as buttons and menus. XUL applications are based on W3C standards including HTML 4.01, CSS 1 and 2, DOM Levels 1 and 2, JavaScript 1.5 (including ECMA-262 Edition 3 (ECMAScript)), and XML 1.0.

The goal of XUL is to build cross-platform applications that can be

ported to all of the operating systems on which Mozilla runs (such as Linux, Windows, Windows CE, and Mac OS X). The layout and appearance of XUL applications are separated from the application definition and logic and the application can be localized for different languages and regions independently of its logic or presentation.

XUL can be complemented by some of the technologies that Mozilla has introduced. The Extensible Binding Language (XBL) [62] is a mark-up language that defines new elements for XUL widgets. Overlays are XUL files used to describe extra content for the UI. Cross Platform Component Object Model (XPCOM) and Cross Platform Connect (XPConnect) have made it possible to integrate external libraries with XUL applications and, finally, Cross Platform Install (XPInstall) provides a way to package XUL application components with an install script [9].

3.1.3   HTML5

HTML5 [40], the successor to HTML 4.01, is currently a work in progress at W3C. HTML5 aims to fix errors and problems with the previous version and add new features especially for web applications. In addition to HTML 4.01, HTML5 includes new versions of XHTML 1 and DOM2 HTML API, which were previously defined in separate specifications.

HTML5 defines two syntaxes, HTML5 syntax and XML syntax, both of which result in a DOM presentation of a document. While earlier versions of HTML were based on Standard Generalized Markup Language (SGML) and used SGML parsing rules, HTML5 has its own parsing rules.

HTML5 introduces a number of new elements and attributes and it also removes many presentational elements and attributes. In addition, some elements will have new semantics [91]. HTML5 is intended to be complemented by CSS and ECMAScript, which are also used in this Thesis.

3.1.4   XAML

Extensible Application Markup Language (XAML) is a user interface mark-up language for Windows Presentation Foundation (WPF) [58], which is a graphics subsystem of the Windows Vista operating system. XAML consists of features from both Microsoft Windows applications and web applications. In WPF, the application UI can be defined with a programming language (such as C#, C++, Visual Basic, etc.) or by XAML [88].

In other words, a developer can use XAML instead of C# or the equivalent to create the UI elements. Using XAML, it is possible to use specialized tools for application development. The tools generate XAML code to run on WPF.

3.1.5 LZX

Laszlo XML (LZX) [65] is the UI description language of the OpenLaszlo platform, used to create web applications. OpenLaszlo is an open source project that consists of LZX and the OpenLaszlo Server, a Java Servlet that compiles LZX applications into either Flash[1] or DHTML depending on the targeted runtime environment [50]. LZX is an XML format that also includes ECMAScript snippets to describe the application logic. In LZX, the UI is described with concrete UI components, which can be tied into a data model.

3.2 Multimodal Technologies

Users of a multimodal system communicate with an application using different modalities (such as hearing, seeing, or speaking). The input can be sequential, simultaneous or composite between modalities, and uses of multimodality can be either *supplementary*, which means that every interaction can be carried out in each modality as if it was the only available modality, or *complementary*, which means that the interactions available to the user vary by modality. [52]

Speech systems can produce efficiency increases of 20-40 percent compared with other interface technologies, such as keyboard input [53]. Although speech interaction seems to be promising, the potential technical problems that they present may irritate users and reduce task performance [85]. To offset these weaknesses, speech is often combined with other modalities [17]. In particular, GUIs are often combined with speech modality.

As a modality, speech is different from a graphical UI, mainly in terms of the navigation within the UI. In the visual modality, more information and navigation options can be represented simultaneously, while speech

---

[1]Flash is a proprietary multimedia platform used to add animation, video, and interactivity to web pages.

interface has to be serialized. On the other hand, spoken command input allows intuitive shortcuts. The following subsections introduce two technologies that combine voice and graphical modalities.

Recently, there have been attempts to bring multimodal interfaces to any web application running on a regular web browser. The main idea is that a speech recognizer and a synthesizer can be added to any web application via ECMAScript. The actual speech processing happens on the server side and the client-side component is just sending and receiving speech and text back and forth. A WAMI toolkit [35] uses a Java applet to access a microphone and speakers of a device. Other example frameworks include an open source project SpeechAPI[2], which uses Flash on the client side, and AT&T's Speech Mashup[3]. Mlakar and Rojc describe a specific wrapper, where web applications are integrated into a multimodal platform and can be accessed on the Web [61].

3.2.1 XHTML+Voice

An XHTML+Voice (X+V) profile [5] is based on W3C standards and is used to create multimodal dialogues with visual and speech modalities. In addition to XHTML Basic and a modularized subset of VoiceXML 2.0 [55], it consists of an XML Events module, the XHTML scripting module, and a module containing a small number of attribute extensions to both XHTML and VoiceXML 2.0. XHTML is used to define the layout of the graphical content of a document, whereas VoiceXML adds voice modality to the document. XML Events provide event listeners and associated event handlers for the profile. Finally, the attribute module facilitates the sharing of multimodal input data between the VoiceXML dialogue and XHTML input and text elements. VoiceXML is integrated into XHTML using DOM events. When an element is activated, a corresponding event handler is called and the event handler synthesizes the text obtained from the element that activated the handler. Also, a speech input to an XHTML form field is assigned through VoiceXML elements. XHTML+Voice documents can be styled both by the visual and aural style sheets.

---

[2]SpeechAPI project, http://www.speechapi.com/

[3]Speech Mashup, http://www.research.att.com/projects/SpeechMashup/index.html

3.2.2 SALT

Speech Application Language Tags (SALT) is an XML-based language for incorporating speech input and output to other languages [98]. The speech interaction model is dialogue-based and it can be embedded, for instance, in XHTML documents. SALT uses a notation of sub-dialogs, which can be added to a page. Although SALT does not have an explicit data model, it binds to external data models, such XHTML forms, using identification (ID) attributes. It has a low-level programming model that allows authors to fully control the speech dialogue interface.

3.3 Communication with the Back-End

This Thesis addresses two aspects of a client's communication with the back-end system. Firstly, it evaluates data serialization models of the approaches introduced in Section 3.1 and, secondly, it proposes a new model to realize server push for web applications.

3.3.1 Data Serialization

For communication between a UI and a back-end system, the user interface state must be serialized for transmission. Having received a serialized reply from the server, it must then be transformed into an application state. Depending on technology, the serialization must be done by the developer separately for each application; otherwise it might be fully automatic.

3.3.2 Server Push

Since HTTP always requires a request from the client side, it is not possible to react immediately to events that occur on the server side of a distributed application. The server-side application must wait until the client contacts the server and then include the change into the response. Such an operation, which is missing from HTTP, is usually called server push, in contrast to client pull. At its simplest, server push on the Web can be emulated on HTTP by polling the server at a certain time interval. This can be done, for instance, by reloading the document periodically or

with Ajax [30] by requesting incremental updates. This causes a lot of additional network traffic, especially reloading the entire document, and there is always a trade-off between the latency and the polling frequency.

Russell coined the term 'Comet' in reference to low-latency data transfer to the browser in 2006 [78]. Comet is not an explicit technology set, but rather a term to describe server-push data streaming functionality. It can be implemented in many ways, such as keeping the connection to a server open, possibly via an *iframe* element, or keeping a connection for XML-HttpRequest [92] object open. The downside of Comet is that the server must keep connections open to all clients it has to update. The central server must also be able to distribute the communication properly [77].

HTML5 specification [40] defines an approach called Server Sent Events (SSE). With SSE, an author can declaratively define a source from which the browser listens for the incoming connections. The communication itself can be realized with Comet. The HTML5 specification is, however, currently still a work in progress at the W3C.

3.4   Requirements

The technologies discussed above are evaluated against the requirements defined in this Section. The requirements are collected from the literature and revised by the Author.

3.4.1   User Interface Languages

Publication V revised requirements for web user interface languages. The requirements were grouped by their sources in the Publication. In this Section, the requirements are regrouped into the UI Definition and UI Logic requirements according to the division in Figure 1.2.

User Interface Definition
The UI definition requirements are from the literature [87, 90] and Publication V. Table 3.1 summarizes the requirement sets.

User Interface Logic
The UI logic requirements are from the literature [90] and Publication V. The requirement sets are summarized in Table 3.2.

Table 3.1: User Interface Definition Requirements

Requirement Sets	Description
General Requirements	Simon et al. discussed the requirements for a generic user interface description format [87], which are device and modality independence and customizability. In other words, the UI representation must be generic and independent of any specific client device technology. The UI description format shall not restrict the modality and the format must provide a high degree of control over layout and graphical appearance.
Separation of Interface Elements from their Presentation	Data and presentation information must be separated [90].
Interface Elements	A UI description must enable automatic UI generation for any target environment and it must support data types [90].
Presentation-related Information	There must be a method for logically grouping the elements of a presentation. The presentation information must include resources such as labels and help text. To allow presentation replacement, it should be possible to share the core UI description between alternative UIs [90].
Paging and Dialogues	A large UI is considered more usable if it is divided into smaller sections. This can be achieved, for instance, with wizard-type paging or with tabs [V].
Repeating constructs	Helps a developer manage large data sets on the UI [V].
Nested constructs	The data sets are typically structured data, such as XML. Nested constructs help when presenting them on the UI [V].

Table 3.2: User Interface Logic Requirements

Requirement Sets	Description
Interface Elements	Possible to define dependencies between interface elements [90].
Run Time and Remote Control	Local computation reduces latency of a network communication. The current state must be explicitly available at run time [90].
Copy-paste	Helps users edit content in an application [V].
Undo-redo	Users must be able to withdraw their actions [V].
Drag-and-drop	Direct manipulation interfaces are considered easy to use [86], [V].

3.4.2 Multimodal Technologies

The multimodal technologies are evaluated based on the W3C Multimodal Interaction Requirements [52]. Table 3.3 lists the requirement sets.

Nichols et al. derived another set of requirements for automatically generating multimodal remote control user interfaces for home appliances and services [45]. Those requirements differ from the W3C requirements. The most notable difference is whether the possibility to author modality-dependent information is explicitly required (W3C [52]) or disallowed (Nichols et al. [45]). On the other hand, one requirement that has been adapted from Nichols et al. is the requirement for two-way communication, which is addressed as part of the communication requirements in Section 3.4.3.

3.4.3 Server Push System

The requirements for communication with the back-end system relate to a server push system. The requirements were specified in Publication VI and are listed in Table 3.4.

Table 3.3: W3C Multimodal Interaction Requirements [52]

Requirement Sets	Description
General Requirements	The integration of modalities should be seamless. It must be easy to author, use, and implement a multimodal language. Requirements also include accessibility, security, and flexible delivery context related issues.
Input Modality Requirements	It must be possible to author modality-related information. The input should be able to be given sequentially, simultaneously, and combined from several modalities. Also, temporal positioning of input events should be available.
Output Media Requirements	Output must be able to be represented both sequentially and simultaneously in different modalities.
Architecture, Integration, and Synchronization points	Compound documents with existing languages should be preferred. Also, the specification should be modular. Data model, presentation layer, and application logic should be separated. The language should provide the means with which to detect or prescribe the available modalities.

3.5 Evaluation

A number of technologies and frameworks exist that are supposed to make web application development easier. This Section analyses the solutions discussed above with the goal of defining the deficiencies in the current solutions.

3.5.1 User Interface Languages

This Subsection evaluates the languages discussed in Subsection 3.1 against the requirements above. The findings in this Subsection are a summary of the evaluations in Publication V. The evaluation was conducted on a three-level scale, using the following levels and corresponding

Table 3.4: Requirements for a Server Push System

Requirement	Description
Protocol	The system must support both push and pull protocols. Based on the literature, HTTP-based web applications benefit from an integration of a push protocol.
Delivery Scheme	The system must support a one-to-many delivery scheme: that is, one that is more precise than broadcasting but not a one-to-one dialogue.
Coherence	The data updates must be delivered in near real time.
Flexibility	The system must support asynchronous communication between loosely coupled components.
Performance	The system must outperform HTTP polling in terms of latency and the amount of transferred data, both of which are considered problems with polling.
Late Binding	The connections between components should be created as late as possible, preferably on runtime.
Ease of Authoring	The application development should not require any new programming language to ease the adoption of the system. Declarative languages are commonly considered easier to author than procedural languages and can even be used by non-programmers. An application should consist of reusable software components.
Web integration	Using existing technologies on the client side eases the adoption of a technology or a framework among the users.

markers:

++ A built-in property. A language supports the property natively.

+ Possible to add or implement with another technology, such as scripts.

− Not possible with the language.

User Interface Definition

The results of the evaluation of the UI definition are shown in Table 3.5. If the UI description language is sufficiently abstract, it is possible to use a single definition to several modalities. With more concrete languages, a separate UI definition must be created for each modality. The definition must be done using separate technology, which doubles the amount of work and complicates the maintenance of the application.

Table 3.5: Evaluation of the user interface definition.

Requirement	XForms	XUL	HTML5	XAML	LZX
General Requirements					
Device Independence	++	+	+	+	+
Modality Independence	++	+	+	+	+
Customizability	+	++	++	++	++
Separation of Interface Elements from Presentation					
Separation of Data/Pres.	++	−	−	−	−
Interface Elements					
Any Target	++	+	+	+	+
Data Types	++	−	++	−	−
Presentation Related Information					
Logical Groupings	++	++	++	++	++
Labels & Help Text	++	+	+	+	+
Presentation Replacement	+	+	+	+	+
Paging & Dialogs	++	++	+	++	+
Repeating constructs	++	+	++	++	++
Nested constructs	++	++	+	++	++

XForms is modality-independent by design and has an abstract UI description that makes it device-independent. XForms uses data types,

which makes it easy to utilize different modalities [III] because a user's input can be handled more specifically. In voice modality, for instance, a grammar-based recognition can be made more accurate. Otherwise, the expected type of input must be defined separately for each modality.

The UI descriptions of other formats do not restrict the selection of devices, but a developer must take into account the target devices and implement the respective UI versions. Although other languages' UI elements can be transferred to other modalities, a lack of data types requires extra work from a developer.

Every format provides control over layout and graphical appearance. In XForms, the UI elements are abstract, whereas the rest of the formats have specific UI elements, which are easier to customize.

The interface elements are not separated from their presentation in any format other than XForms. In XForms, the data model can be accessed through a separate binding layer. XForms is also easier to use in any target since its UI description is more abstract. HTML5 and XForms support data types, whereas others do not.

The presentation can be grouped well with all languages. XForms provides an explicit way to include labels and help texts, while they can be realized in other formats with normal text. It is possible to provide an alternative presentation with all formats.

XForms and XAML natively support repeating structures, paging, and dialogues. XUL also supports paging. With other formats, the implementations require some script programming. Nested constructs are supported by all formats. XUL, XAML, and LZX provide it natively, as does XForms if a proposed XForms tree module is used [71]. HTML5 requires scripting to complete the nested constructs.

User Interface Logic

The results of the evaluation of the UI logic are summarized in Table 3.6. The interface elements can have dependencies in all formats, but only in XForms and XAML this is a built-in property; in other formats, the dependencies must be realized through scripts. Local computations (such as data validation) can be realized with scripts in all formats. In addition, XForms provide native features for local computation. The state of the UI can be stored in a data instance in XForms, XAML, and LZX. LZX has some flaws when using multiple pointers for a single data instance. With these languages, the state of the UI is always available. In XUL and

HTML5, the state must be stored and queried separately. These differences are discussed in more detail in Subsection 3.5.3.

Table 3.6: Evaluation of the user interface logic.

Requirement	XForms	XUL	HTML5	XAML	LZX
Interface Elements					
Dependencies	++	+	+	++	+
Run Time and Remote Control					
Local Computation	++	+	+	+	+
Serialization	++	+	+	++	++
Synchronization	+	+	+	+	+
Copy-paste	+	+	++	++	++
Undo-redo	+	+	++	++	+
Drag-and-drop	+	++	++	++	++

Copy-paste can be used with all the formats. HTML5, XAML, and LZX support it natively (only for text in LZX), whereas it can be implemented in XForms and XUL. Undo-redo and drag-and-drop have native support in HTML5 and also in XAML with certain elements, while they can be implemented in XForms, XUL and LZX. With XForms, a developer must alter the data model accordingly, which makes it harder than with the other two. In short, XAML is the best at handling typical interaction patterns, although all of the formats handle these requirements well, even XForms, which is the most abstract format considered here.

3.5.2 Multimodal Technologies

This subsection evaluates XHTML+Voice and SALT. In order to support supplementary use of modalities, both speech and graphical modalities must be implemented separately with both X+V and SALT. The complementary interaction is more natural for both of these technologies. Both SALT and X+V allow seamless synchronization of modalities at all levels, but they require scripting to enable it. Neither technology has multilingual support. Moreover, accessibility is a well-known problem of XHTML. Neither technology fulfills the Ease-of-authoring requirement well because modalities must be implemented separately. On the other hand, the Ease-of-use is good because of the greater expressive power for modality.

Neither language meets the Flexible Delivery Context requirement to any great degree. There may be additional tools on top of the languages that would transform the document based on the context. In order for this approach to be successful, the abstraction level should be high. X+V has a higher abstraction level than SALT. For non-automatic (that is, authored) approaches, a solution would be a set of DOM events to notify context changes. For Navigation Specification, both specifications support *navindex*, or similar, for the visual part. SALT and X+V support detailed navigation specification for aural modality.

In input processing, X+V and SALT both rely on modality-dependent strategies, such as voice grammars. This means that the author can easily control input processing in each modality in X+V and SALT, although they are hard to author and maintain. Both languages support sequential, simultaneous, and composite multimodal input well. Neither approach caters for Semantics of Input because they do not have data types or input types. In addition, both technologies lack support for Coordinated Constraints between modalities. Both languages meet output media requirements similarly, although SALT provides better synchronization of output events by using a special Prompt Queue. Finally, neither language properly supports the Separation of Data model, the presentation layer and application logic, whereas they do support Synchronization of Granularities.

3.5.3   Communication with the Back-End

Currently, as mentioned in Subsection 3.3.2, server push can only be realized by emulating it using HTTP. Thus, the push system requirements from Subsection 3.4.3 are not used to evaluate any specific HTTP-based solution in this section. The requirements are used to formulate a new server push system in the next section. Instead, this section presents a comparison of a data serialization model of different UI description languages.

The data serialization models are compared from the technologies whose UI definition and logic characteristics were evaluated in the Subsection 3.5.1. The data serialization is automatic in XForms (cf. Figure 3.1a), since the data model is a live XML document object model, which is automatically serialized and de-serialized. In addition, the submission is automated.
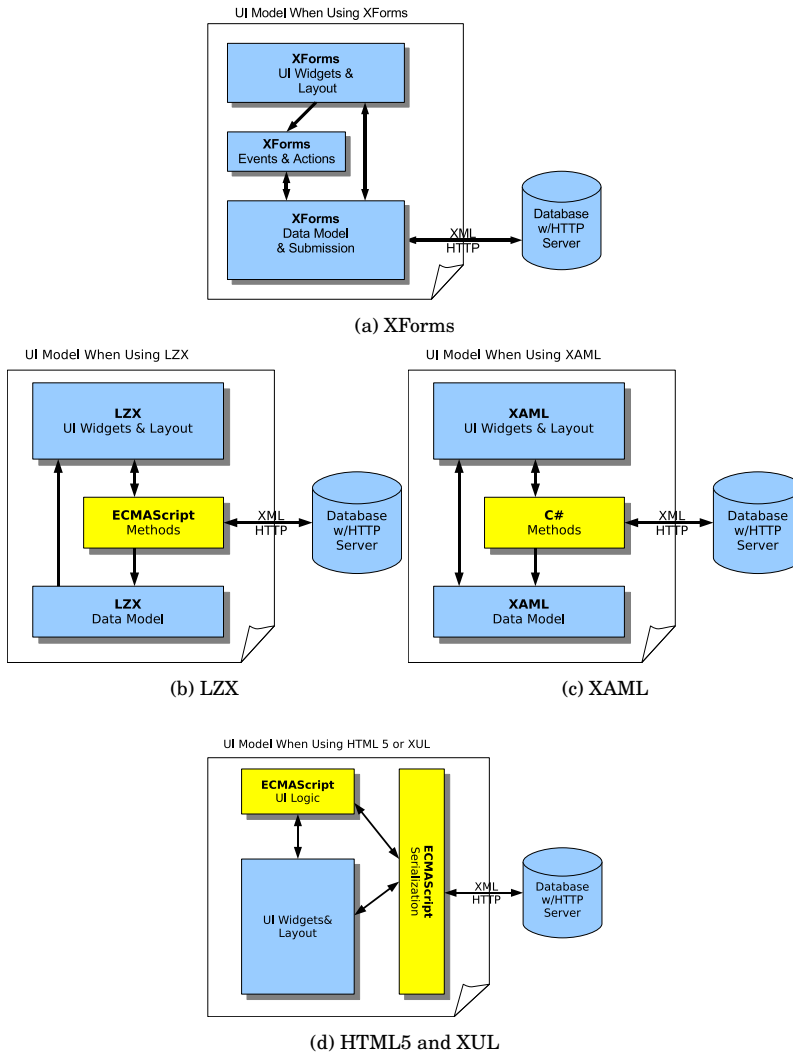
(a) XForms

(b) LZX
(c) XAML

(d) HTML5 and XUL

Figure 3.1: Communication between the UI and the back-end system.

LZX and XAML also have data models, which means that, while seri-alization is automatic, the submission must be realized through a pro-gramming language. In addition, in LZX an author must implement logic in order to keep the data model up-to-date (see Figure 3.1b). The data model provides data for UI but the UI cannot automatically update the data model; that must be implemented separately for each case. The data model updates in XAML (cf. Figure 3.1c) are automatic; otherwise, it is similar to the LZX model.

XUL and HTML5 have no explicit data models, and communication be-

tween a back-end process and the user interface must be re-implemented using ECMAScript for each user interface, as shown in Figure 3.1d. For example, when the server sends back updated structured content, there must be a script that updates the corresponding DOM. This means that authoring and maintaining XUL- or HTML5-based applications is more complicated than it is for others.

In summary, the more a developer has to implement logic to serialize and utilize the data, the more difficult the whole development becomes. The automatic serialization and dynamic bindings of XForms makes it the easiest format with which to build a UI, while the use of XAML and LZX data models is partially automated. HTML5 and XUL require the most work in terms of using and serializing the UI data of an application. All of the logic for handling the data must be implemented separately for each application.

3.6   Summary

The evaluation of the UI languages was based on two sets of requirements: UI Definition and UI Logic. A comparison was also performed of the languages' communication with the back-end. The comparison revealed that XForms was the best format because it is superior in terms of UI Definition requirements, whereas other languages are missing several required features. XAML best meets the UI Logic requirements, although the difference between XAML and HTML5, LZX or XForms is not significant. XUL was found to be the worst in this category. Finally, XForms' data serialization model is the best among the languages, whereas serialization is most difficult with XUL and HTML5.

Both multimodal technologies are based on a model in which all the modalities are implemented separately. This allows specific implementations of each modality, but, naturally, requires more work and, in particular, makes it more difficult to maintain applications. In short, even though it is possible to implement multimodal applications with both technologies, they do not conform very well to the requirements.

4 Proposed Improvements

Based on the comparison of common XML-based languages, this Thesis uses XForms combined with XHTML as a UI description technology. XForms was selected because its compliance with the requirements means that it can provide both UI definition and UI logic, and, in theory, it should make it easy to create multimodal applications. This Chapter starts by introducing the design principles of this model. The following Sections discuss a model for creating multimodal applications with XForms and a model for implementing push updates for web applications. Finally, a model for delivering push updates to XForms UI definition is presented.

4.1 Design Principles

The improvements are defined with certain design principles in mind. These principles have been formed by taking into account the requirements and the evaluation in the previous Chapter and also the problems defined in Section 2.3.

Use declarative descriptions wherever possible. Declarative technologies automatically define most used operations, which means that authors do not have to repeatedly define them. A declarative description can be analyzed automatically, which allows the use of graphical editors, for example.

Use minimum number of technologies. Limiting the number of required technologies helps simplify application development. The goal is to extend existing technologies rather than invent new ones.

4.2 Multimodal Interaction

In terms of multimodality, the main advantages of XForms are data typing, accessibility, and less reliance on scripting. The accessibility is due to the high semantic level of the elements. For instance, a mandatory label on groups and form controls is better than a heuristic search of a possible label. Similarly, datatypes can be utilized well to generate speech grammars. In addition, XForms supports dynamic changes in the UI based on user input. Publication III introduces a model of how XForms can be used as a multimodal authoring format. The author of this Thesis has assisted Mikko Honkala in designing the system.

The approach, called XFormsMM, combines XForms as a *Multimodal Application Authoring Format* with *Modality-Dependent Stylesheets* and a description of a *Multimodal Interaction Manager*. Figure 4.1 depicts the execution environment. The interaction manager interacts with the DOM of the XFormsMM document and synchronizes the Visual and Aural Renderings so that simultaneous multimodal input and output and flexible modality switching are possible.
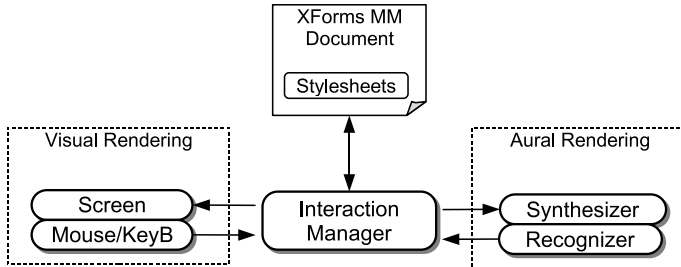

Figure 4.1: XFormsMM Execution Environment.

XFormsMM user interfaces are authored in the XForms 1.0 language. This means that non-speech-enabled user agents will be able to present the user interface in the visual modality. The modality-independent portion includes the XForms Data Model and the abstract UI. The data model is further separated into instance data and constraints, while the UI contains grouping and abstract form controls. XHTML 1.0 is used as the host language. The main modules used from XHTML are: Structure, Stylesheet, Hypertext, and Image Modules, while the Forms module is replaced by XForms. The modality-dependent portion contains stylesheets.

Supplementary use of modalities [52] means that every interaction can be carried out in each modality as if it was the only available modality.

This is provided in XFormsMM by default. Every user interaction is defined using the XForms User Interface Module [21], and can interact with either speech or GUI. On the other hand, *complementary use of modalities* [52] means that the interactions available to the user differ from one modality to the next. In XFormsMM, this is provided by the use of modality-dependent CSS.

The main roles of the XFormsMM interaction manager are synchronization of the modalities, flexible switching between the modalities, and navigation. The main idea behind the interaction manager is that aural and visual states are synchronized automatically. Setting the focus in one modality automatically transfers the focus to the other modality. Since the modalities share a single XForms data model, all data items and their constraints are also shared automatically.

Each modality has its own rendering subsystem, the main roles of which are to output the current state to the user and to receive modality-dependent events from the user. The rendering takes care of detailed events (such as mouse movements in visual modality or partial speech recognition events in aural modality), and provides higher-level events (such as value changed and focus changed) to the interaction manager.

4.3   Server Push System

This section discusses a server push system that can be used for web applications. It is designed for web applications that have to submit changing data to users as soon as the data has changed. Such applications include auctions, score services, and stock quote services. The design of the server push system is based on the requirements defined in Subsection 3.4.3. The system uses a publish/subscribe (pub/sub) messaging model.

4.3.1   Publish/Subscribe

Publish/Subscribe is an asynchronous messaging model. Publishers send their messages to channels rather than directly to subscribers, who receive messages from channels to which they have subscribed. This makes the paradigm loosely coupled. In fact, pub/sub can be decoupled on three dimensions: space, time, and synchronization [24]. In other words,

senders and receivers do not have to know each other; the messages can wait on a channel for subscription, and the production and consumption of events do not prevent other activity at either end of the system. The removal of dependencies between parties means that the paradigm adapts well in distributed environments, which are asynchronous by nature [43]. The pub/sub paradigm is an example of the push system as the publishers push data to the subscribers through the channels.

Subscription of the pub/sub system can be channel-, content- or event-based [24]. In channel-based systems, a receiver subscribes to a certain channel, from which it receives all the published messages. In the content-based pub/sub model, a subscriber defines filters for the subscription and it will receive messages that pass the filters. Event-based systems are founded on event types, to which receivers subscribe. In addition, the pub/sub system can be a combination of the above-mentioned types.

4.3.2 Communication Paradigm

The communication paradigm combines both push and pull protocols. That kind of combination has been proposed by Deolasee et al. [19] and Hauswirth and Jazayeri [36]. This Thesis proposes concrete components and their communication for the combined push and pull communication.

A user agent fetches a document from the web server using HTTP protocol. When the web application is running on the user agent, dynamic data is pushed to the client whenever necessary. In addition, the model allows for a graceful degradation if the push component fails. The system can rely on pull-based polling as a backup mechanism.

The idea is to use the pub/sub-messaging paradigm with a combined channel and content-based model, in which user agents subscribe to the channels according to their current state. In the case of web applications, for example, the user agents subscribe to the channels according to the location of their current resource. In other words, there is a one-to-one match between URLs on the web server and the channels on the push server.

The Web Server communicates the correct channel to the user agent within the data sent over HTTP. Accordingly, it can be said that the model supports the Representational State Transfer (REST) paradigm on which the Web relies. There is an Event Source for every channel on the Push

Server. The Event Sources track the changes on the server side, create update events according to the changes, and publish them on their channels. The Push Server distributes the updates to every user agent that has subscribed to the corresponding channel. This is known as a multicast delivery scheme. A more specific distribution can be achieved by adding content-based subscription. In this manner, only relevant updates are filtered to each subscriber.

By way of summary, Figure 4.2 shows the communication between components of the system. The user agent fetches a document from the Web Server via HTTP, as usual (1), but, in addition, it also receives the address and subscription details of the Push Server. The user agent uses this data to connect to the Push Server and subscribe to the relevant channel (2). In response, the Push Server submits the latest message of the channel to keep the user agent up-to-date (3). This ensures that any possible change that occurs between the Web Server's response and the subscription is delivered to the client.
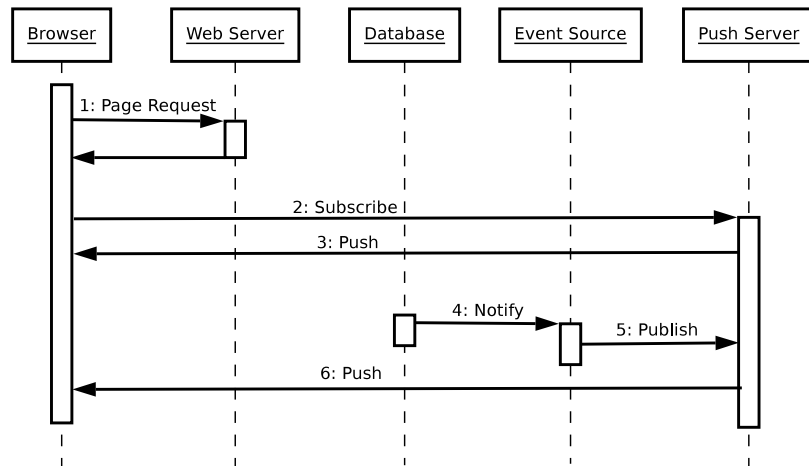

Figure 4.2: The communication between components of the system.

When a data is modified, for instance in the database, a trigger notifies the Event Source (4). The Event Source creates an update event and publishes it on the Push Server in the appropriate channel (5). The Push Server pushes the event to all subscribers of the channel (6).

4.4 Remote DOM Events

The push update consists of three operations, which are mostly independent from each other and even their order may vary depending on the used technologies [IV]. The operations are:

Server Push. An update must be pushed to the client whenever it occurs on a system.

Targeting the update. When an update arrives on client-side, there must be a way to target the update for the correct elements in the existing document.

Adding the UI semantics. By default, a system provides its updates in the form of raw data, which must be transformed into UI declarations. This can be done on both the server and client sides.

The previous Section presented a system that supports server push; this section discusses the remaining two operations. The purpose of the server push is to modify the DOM tree on the client side, which can be done via ECMAScript or defined declaratively. The declarative methods make it possible to define the target and modification on the server side. With ECMAScript, it is done on the client side. The modification methods are discussed in Publication IV. W3C has produced a working draft of Remote Events for XML (REX) [6], but, at present, has stopped the specification work because of patent issues [97]. In spite of this, REX represents the declarative concept to describe the update events. This Thesis refers to that concept as Remote DOM Events (RDE).

This thesis represents a declarative model to remotely modify the UI. The model is based on the RDE concept and XForms. The benefit of XForms is that the data of a form is separated from its presentation. In other words, the data can exist in same form in both the client and the server side. The outfit of the data is defined in an XForms form. In order to update the XForms instance data with RDE events, the RDE interpreter must update the instance document instead of the UI document. Figure 4.3 presents the relationship between XForms instance data and the actual UI DOM. As the figure shows, the instance is separate from the UI. There is a two-way connection between an instance node and its UI node.
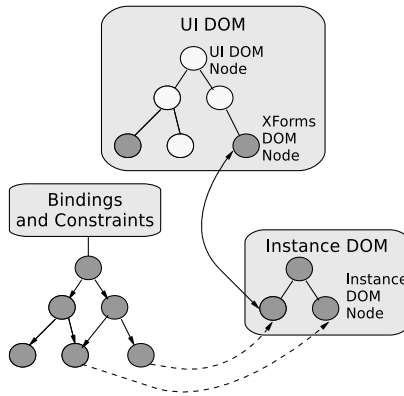
Figure 4.3: The XForms instance in relation to the UI document and the form constraints.

XForms can also be used to automatically evaluate the content of the update. The XForms *bind* element can retrieve the content and make calculations on it or compare its value to other values in the instance. The result can also be made visible on the UI. Figure 4.3 also depicts a dependency graph through which the XForms engine controls the values of the data instance. The bind elements affect the graph among others.

5 Prototype Implementations

As a proof-of-concept, software that supports the selected UI description model and the server push system has been implemented. The goals of the implementations are to show the feasibility of the proposed improvements and to find their possible implementation issues. Firstly, the components of an XML user agent are discussed and, secondly, an implementation of the proposed server push system.

5.1 XML User Agent Components

The Author of this Thesis has participated in the development of the X-Smiles XML browser [95], which supports several XML vocabularies and allows combinations of them [70]. The main components of the X-Smiles browser can be divided into four groups: XML processing, Browser Core Functionality, Markup Language Functional Components (MLFCs) & ECMAScript Interpreter, and User Interfaces. These are shown in Figure 5.1. XML Processing converts the XML documents into a DOM tree. The core of the browser controls the overall operation of the browser, which includes browser configuration, event handling, content management, etc. MLFCs create XML language-specific DOM elements and render the documents using specific GUI bindings, which further use a component factory.

The author has co-implemented the visual and aural rendering and XHTML and XForms interaction with Mikko Honkala for the browser. This section describes the implementation of the CSS Layout Engine, its integration into X-Smiles' XForms component, multimodal interaction, and RDE.
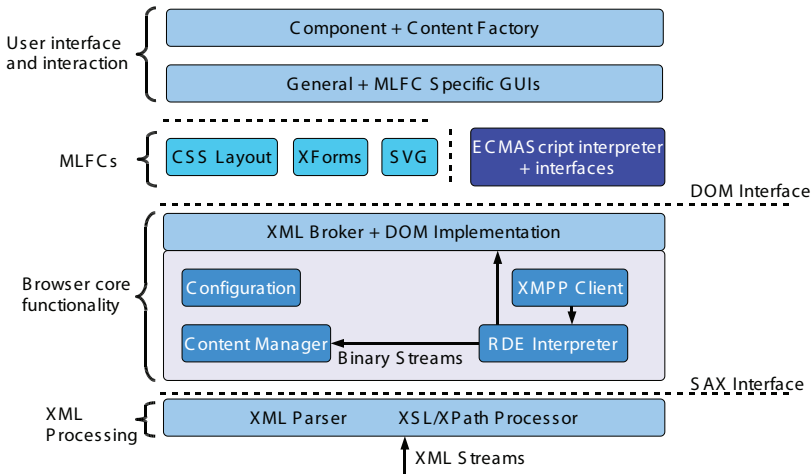
Figure 5.1: Architecture of the X-Smiles XML browser (adapted from [95]).

5.1.1 CSS Layout Engine

The CSS Layout Engine provides visual rendering of the web applications. Publication II presents requirements and implementation of the CSS Layout Engine. The Layout Engine consists of a renderer and a set of views. It also cooperates with a CSS style context, which provides the styling attributes to the elements, and a DOM document. The CSS Layout Engine is one of the MLFCs of X-Smiles (cf. Figure 5.1) and it processes the CSS styled documents in X-Smiles.

The general idea of the layout engine's operation is as follows. The CSS Layout module creates the layout engine and sets a document to be rendered. The document is accessed through the DOM interface by the layout engine (cf. Figure 5.2). The views are created according to the DOM elements and every view creates associated child views. When the views have been created, they are laid out and painted. The process is repeated partially due to certain operations (e.g., scrolling the document, resizing the window, or modifying the content).

There are three kinds of views. The content-specific views, such as the image and the text, keep up the spatial information and take care of the rendering of the content. The image view also fetches the image data from the source. The CSS-specific views, such as the block and the inline, embed the content views and instruct the laying out according to the CSS specification. Finally, there are additional views to structure the content.
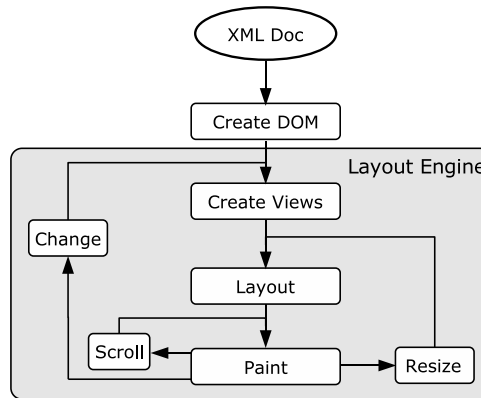
Figure 5.2: Flowchart of the layout engine.

For instance, when a paragraph is laid out, the text must be divided into rows; this is done through the paragraph and the row views, as Figure 5.3 shows. On the left-hand side of the figure is a snippet of HTML code including a *p* element, which embeds text and an *a* element. A DOM tree of the markup is represented in the middle of the figure. The # characters correspond to the text nodes in the code. The view tree created by the CSS layout engine is depicted on the right-hand side of the figures and there are pointers that show which DOM elements correspond to which views. Additional views are used to lay out the document properly. As can be seen from the final layout in the bottom left corner, there are two rows and the inline element (*a*) has been divided between them.

User Interface Modifications

The DOM document can be modified after its creation by, for example, Remote DOM Events or scripts. Obviously, these modifications must be delivered to the layout engine. The modifications are caught by the extended DOM elements. If something happens to an element, it updates its style and also those of its children because some of the CSS properties are inherited by the descendants. The view tree is modified according to the changes.

If the modifications have an effect only on the layout of the element in question, then the corresponding view tree is created, laid out, and, if on the screen, painted again (cf. Figure 5.2, the *Change* operation). The process is equivalent to Mozilla's incremental layout [99]. However, if the changes also affect on the other elements (such as the size changes), then the entire affected view tree must be laid out and painted again, although
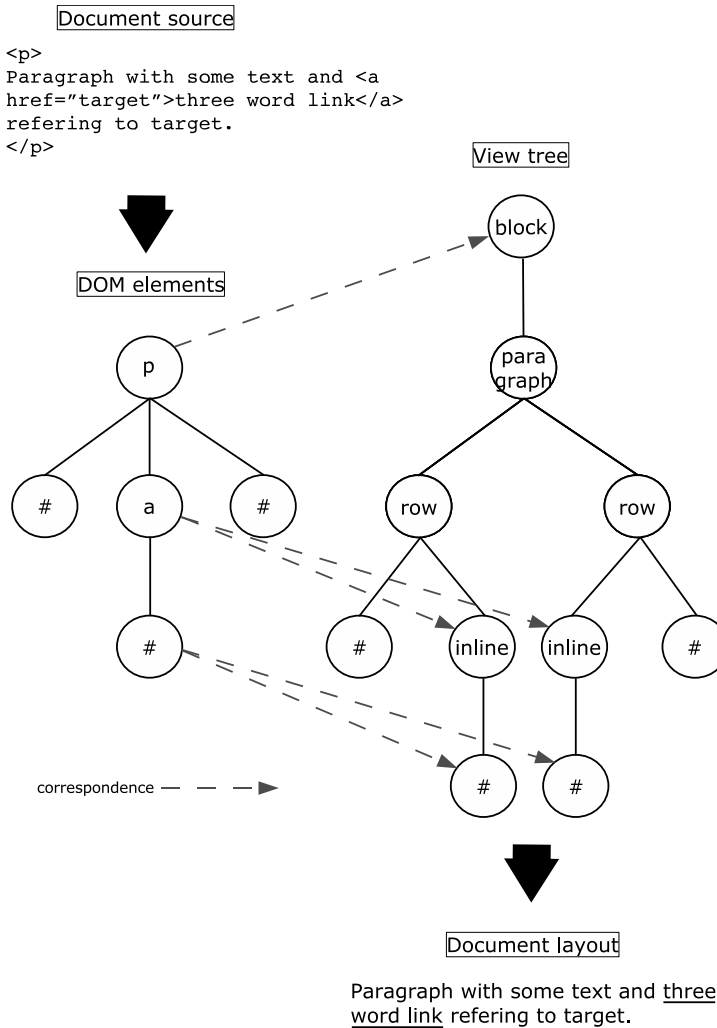
Document source

```
<p>
Paragraph with some text and <a
href="target">three word link</a>
refering to target.
</p>
```

View tree

DOM elements

correspondence — — — ➤

Document layout

Paragraph with some text and <u>three word link</u> refering to target.

Figure 5.3: Illustration of the text layout.

not created (cf. Figure 5.2, the *Resize* operation).

XHTML+XForms

The CSS layout engine supports the rendering of compound documents. The formats of a compound document do not all have to follow the CSS layout model. In those cases, the layout engine provides a region in which a component supporting a non-CSS format can render its content. The content can be a single element (such as XForms) or an entire document (such as SVG). Publication I introduces XHTML 2.0 implementation, which utilizes the layout engine to render the XHTML and XForms modules of the

XHTML 2.0 specification. The implementation shows the feasibility of the model of the layout engine. However, because of the design of the XForms language, integrating it with the CSS layout engine is not a completely simple matter. For instance, there is a notion of a cursor inside a repeated construct, which has no correspondence in the DOM. Furthermore, the element describing the form control contains a label element inside it, making it difficult to style just the form control itself [I]. XForms uses pseudo classes and elements to solve those issues that had to be implemented in order to support the compound.

5.1.2 Multimodal Interaction

The layout engine is also part of the multimodal interaction framework of the X-Smiles browser. Along with the XForms component, it provides the GUI rendering and the UI functionality. Publication III discusses the framework and its implementation and Figure 5.4 illustrates the architecture of the implementation.
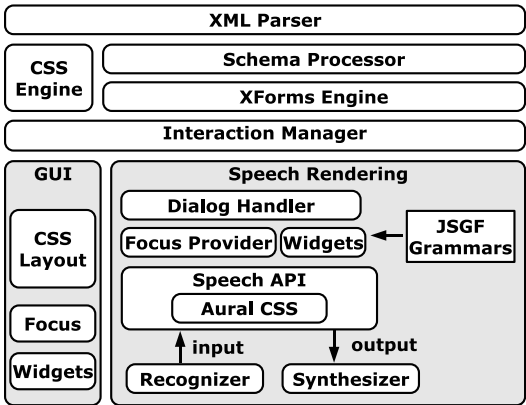

Figure 5.4: The components of the XFormsMM implementation in X-Smiles.

The XForms engine [41] provides the state of the document. Interaction manager is a thin layer that communicates between the XForms engine and the GUI and speech rendering subsystems. Speech rendering consists of a *dialogue handler*, *speech widgets*, a *focus provider*, and a *Speech API*. The dialogue handler uses speech widgets and a focus provider to provide speech UI navigation and form field filling. Speech widgets are created for each element according to the datatypes, and they store the input data to a form in the correct format. They are the aural counterparts of the visual

form controls, while the visual focus management is already provided by the graphical XForms implementation. As a result, the interaction manager registers itself as a listener for graphical focus events. In addition, a speech focus provider was created. The Speech API is a low-level interface for speech recognition and synthesis.

5.1.3 Remote DOM Events

The CSS layout engine supports the RDE concept described in Section 4.4. An RDE Interpreter (cf. Figure 5.1) receives update events for instance from an Extensible Messaging and Presence Protocol (XMPP) client as a binary stream. The next section discusses a server push system that uses XMPP. The binary streams are forwarded to a Content Manager, which returns the XML markup back to the interpreter, which parses the markup and creates a mutation event based on the content of the markup. The mutation event is forwarded to the DOM implementation of X-Smiles, which further modifies the live DOM of a document to be edited. Publication IV describes four methods for utilizing RDE in web applications. This Thesis uses the XForms+RDE method as described in Section 4.4.

5.2 Server Push System

This section discusses an implementation of the Push System that is based on the XMPP protocol and conforms to the web application communication paradigm. The main reasons for selecting XMPP were its wide usage in other XML streaming applications, its pure push nature, existing implementations, and the possibility of using it with legacy browsers and firewalls.

5.2.1 XMPP

XMPP is a protocol for streaming Extensible Markup Language (XML) elements in near real time between any network endpoints [79]. It was initially developed as a protocol for Instant Messaging (IM) applications [80]. The XMPP technology set comprises of core functionality and numerous extensions. The core defines how XML snippets are handled for instance in IM applications and the extensions utilize that base. Among the exten-

sions are those for XMPP publish/subscribe [60] and HTTP binding for XMPP communications called Bidirectional-streams Over Synchronous HTTP (BOSH) [66], which can be used if regular XMPP communication is blocked (by a firewall, for example).

BOSH uses Comet to mimic two-way communication between the Server and the browser. However, even if it emulates push and is therefore no longer pure XMPP, it has many benefits compared to plain Comet given that it has other characteristics of XMPP. These include persistent connections even if the underlying network connection is unreliable, a prescribed messaging model and format, and authentication. XMPP BOSH can be said to provide a standardized communication model for Comet.

5.2.2 Components

Figure 5.5 depicts the components of the system. All of the software is open source and extended as needed. The Web Server is an Apache Tomcat Servlet container[1] and the database is an eXist-db XML database[2]. There is also an XMPP Server and Event Sources in the server-side. The XMPP Server is an Openfire XMPP Server[3] that has native support for XMPP pub/sub. Event Sources are system-specific components that use the Smack XMPP client library[4] with su-smack[5] pub/sub extension for XMPP communication.
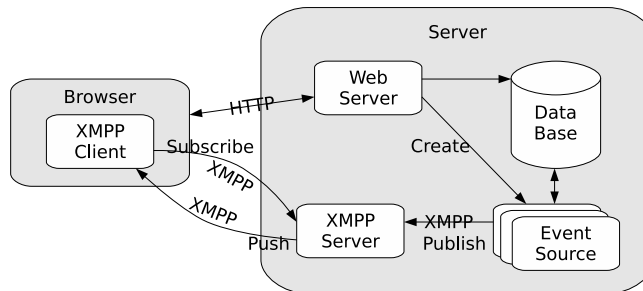

Figure 5.5: The components of the Push System.

User Agents of the system include a browser, the XMPP client, and an

---

[1]Apache Tomcat, http://tomcat.apache.org/

[2]eXist-db, http://exist.sourceforge.net/

[3] Openfire XMPP Server, http://www.igniterealtime.org/ projects/openfire/index.jsp

[4]Smack API, http://www.igniterealtime.org/projects/ smack/index.jsp

[5]su-smack, http://static.devel.it.su.se/su-smack/

update handler. The XMPP Client can have native integration into the browser or it can be an ECMAScript library, which comes with a web application. This means that the system can be used with legacy browsers. As a proof of concept, both native and ECMAScript implementations were performed. In the native implementation, the XMPP components were integrated into X-Smiles [95], which uses the same libraries as the Event Sources for XMPP communication. The update handler depends on the format of the updates, which will be discussed below.

The ECMAScript implementation uses the Strophe XMPP library[6], which was extended to support XMPP pub/sub. There is also an ECMAScript implementation of the update handler. The Strophe pub/sub implementation provides a generic interface that creates and parses the XML stanzas used for XMPP pub/sub communication. Although the client must still handle the connection, but the library creates the content of the messages.

5.2.3 Operation

Figure 5.6 describes the operation of the system. The database has a mechanism that triggers an event when a collection in a database is modified. The Event Sources have registered themselves to the relevant triggers. The triggers notify registered Event Sources when the database has been modified (1). The Event Sources publish the modified information on their channels in the XMPP Server (2), which further pushes the event to the subscribers (3). The browser handles the event and updates the relevant node in a document (4).
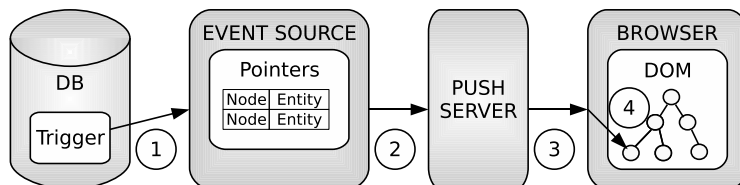

Figure 5.6: Conceptual model of the operation.

---

[6]Strophe, http://code.stanziq.com/strophe/

6 Discussion

The results of this Thesis can partly be utilized in a general sense, while they have also partly been tied to certain technologies. In particular, the proposed communication paradigm of the server push and comparison of UI description languages can be of general interest. The improvements of the UI description and logic are intentionally tied to XForms technology, since this was found to be the best format in the aforementioned comparison. The following Sections evaluate the results of this Thesis.

6.1 User Interface Languages

The requirement conformance reveals what types of applications the languages suit best. XForms copes best with the universal interaction requirements, but is weakest with typical interaction patterns, which, in turn, are best fulfilled by languages targeted to application UIs. XForms' main targets are complicated forms, which means that it simplifies the management of large data set on the client side. XForms also excels when accessibility requirements are involved. XForms' abstract UI description makes it easier to develop applications for different modalities and devices. Customizability is important when using an application in different contexts [49], and is easiest in languages with specific UI elements, including XUL, HTML5, XAML, and LZX. Finally, a desktop-like UI with direct manipulation is best achieved with XAML.

The various requirements are somewhat at variance with each other. For instance, conforming well to technical requirements may make it difficult to implement all the typical interaction patterns. For example, realizing drag-and-drop with XForms' UI independence is rather complicated.

XForms was selected as the UI language of this Thesis because of its technical merits. As predefined in the Introduction, this Thesis did not attempt to evaluate what would be required to adopt a new technology

on a large scale. Either way, the adoption would be problematic since browser vendors cannot stop to support legacy content on the Web; this is evidenced by the vendors' strong support for HTML5 at W3C. As the comparison shows, although an incremental development of HTML might be sufficient at the present time, it remains to be seen whether there is a need to abandon backward compatibility and make a technology leap in the future. There is nothing that would have prevented to implement the proposed improvements with another technology than XForms, but it would have required more work. Depending on technology, other formats lack data types, abstract UI description, or a data model. Those all make it easier to apply the proposed improvements to XForms.

6.2   Multimodal Interaction

Publication III evaluated the XFormsMM approach against W3C multimodal requirements and compared X+V and SALT. XFormsMM supports supplementary interaction well, since all interaction is always automatically available in both modalities, and the user can, at any point, switch between the modalities. Synchronization of modalities is seamless at the field level with XFormsMM. It also has clear benefits over the others in terms of both multilingual support and accessibility. Ease-of-authoring is best in XFormsMM, where only single UI definition is required, compared to authoring both aural and visual parts in SALT or X+V. On the other hand, Ease-of-use, when using only aural modality, is better in SALT and X+V, since they have greater aural expressive power.

In Input processing, X+V and SALT both rely on modality-dependent strategies, such as voice grammars, while XFormsMM uses cross-modality definitions, such as data and input types. This means that an author has more control on input processing in X+V and SALT, although XFormsMM is much easier to author and maintain. XFormsMM, with data- and input types, is best able to provide semantics of input. In addition, coordinated constraints are best provided by XFormsMM, by sharing a single structured data model (along with declarative constraints and calculations) between modalities.

Only XFormsMM properly supports separation of the data model, the presentation layer and the application logic. SALT and X+V best support synchronization granularities, since XFormsMM only supports field- and

event-level synchronization.

The XFormsMM and X+V were also compared by implementing similar application with both of the technologies [III]. The XFormsMM implementation was notably smaller in code size, even though it has more features. This is mainly because X+V implementation also contains grammars, which are generated automatically in XFormsMM; in X+V, the UI must be partially defined twice, that is, for both modalities separately, and the X+V implementation requires a lot of scripts.

In short, the main difference between XFormsMM and the other two is the semantic level, which is higher in XFormsMM. This results in better ease-of-authoring, since only one UI description needs to be written. Also, synchronization between the modalities is automatic in XFormsMM, while scripting is not required for most user interfaces.

6.3   XML User Agent Components

The implementations of the XML User Agent components demonstrate the feasibility of the XHTML + XForms compound, remote DOM events, and multimodal interaction with XForms. The implementation of the CSS layout engine demonstrates how to support compound documents. It can also include non-CSS based markup either as single elements or whole documents [I, II].

This Thesis uses the remote DOM events paradigm with XForms; that is, the remote events modifies XForms' DOM in the client side. It is easier to maintain the UI description with the methods in which UI description resides only on the client side. Additionally, the RDE+XForms method provides a means with which to automatically react to the content of the update. The RDE+XForms can be used with any UI description language as long as it is in the XML format, presuming that the format can be combined with XForms.

6.4   Server Push System

The implementation of the server push system shows that it is possible to implement a system that pushes database modifications all the way to the web browser in near real time. Overall, the system fulfills the require-

ments well. XMPP, with its pub/sub extension, suits this kind of communication. Decoupling between the core communicators, that is, the user agents and Event Sources, has been successful. Both sides communicate using a standard XMPP Server and do not have any inter-dependencies. In addition, the bindings between the components are formed on the runtime.

The model provides a means with which to fulfill the coherence requirement, although the current experimental implementation requires an optimization on database triggering in order to conform to the requirement. The performance of push communication is similar to Comet, which has been shown to outperform legacy HTTP polling applications [2, 12]. In addition, the XMPP-based system has some benefits over plain Comet, as discussed in Subsection 5.2.1.

A new protocol typically requires new software on both the client and server sides, while developers must learn the new protocol. Although it is not necessarily difficult for developers to implement the requirement regarding new client-side software, it has to be installed by most of the end-users before application developers will actually start to use it. The update cycle on users' browsers is slow. If the same functionality can be offered via ECMAScript, it can be adopted immediately. This Thesis has shown that the XMPP communication model can be applied immediately on the Web, even though it has to be bound to HTTP. XMPP with HTTP binding could be used as a kind of standardized syntax for Comet communication, which could even help developers to create Comet-based applications. Furthermore, the application would automatically work with user agent with XMPP support.

7 Conclusions

This Thesis has proposed improvements to web application user interface technologies. The work was conducted by answering the research questions formulated in Section 1.3 (Q1-Q4). This Section summarizes the contribution of the Thesis and defines the future work items.

7.1 Contribution

The work was outlined to cover novel and advanced web technologies that are expected to improve the state of the art. Furthermore, the scope included web application UI technologies and UI communication with a back-end. The Thesis represents a large set of requirements within this scope, which were collected from the literature and complemented by the author.

The evaluation of the UI languages was based on two set of requirements: UI Definition and UI Logic. In addition, a comparison was made of the languages' communication with the back-end. The result of the comparison was that XForms was the best format overall. It exceeded UI Definition requirements, whereas other languages were missing several required features. Additionally, XForms' data serialization model was the best among the languages. The UI Logic requirements were best fulfilled by XAML.

Multimodal technologies, X+V and SALT, were evaluated against W3C multimodal requirements. They are both based on a model in which the modalities are, mainly, implemented separately. This allows specific implementations of each modality, but, naturally, requires more work and, in particular, makes it difficult to maintain the applications. In short, even though it is possible to implement multimodal applications with both technologies, they do not conform well to the requirements.

Improvements were proposed on the basis of both the requirements and

the evaluation. Firstly, design principles were defined to ensure consistency of the improvements. The aim was to mainly use declarative descriptions and, based on the evaluation, specifically XForms. Moreover, the goal was to use only a small number of technologies and extend the existing ones rather than defining new ones.

The proposed improvements relate to multimodal interaction, server push, and declaratively defined UI updates. Firstly, XFormsMM specifies how to use XForms in order to create multimodal applications. The framework fulfilled the requirements better than the reference technologies and made it easier to implement a use case. Secondly, a model and an implementation of improved web application communication were introduced. The system uses XMPP to push information to clients. Finally, a method for dynamically updating the UI using XForms was discussed. The method, which can be integrated with the server push system, defines UI modifications caused by the updates declaratively.

In addition, the proposals have been implemented as a proof of concept and they were implemented into the X-Smiles XML browser. In order to fully adhere to the design principles, the browser also required the implementation of other components, including a CSS layout engine with compound document support.

Section 2.3 highlighted the problems with the current web technologies and this Thesis has proposed solutions for two of them: multimodal interaction and server push. It is notable that selecting XForms as the UI description language solved the other problems.

7.2 Future Work

According to the evaluation of the user interface description languages, XForms' major flaw is its lack of support for direct manipulation, which includes UI operations such as drag-and-drop. This flaw is mainly due to the abstract UI description and use of separate data model [V]. Nevertheless, it should be investigated how direct manipulation methods could be added to the XForms natively. Although it is possible using an ECMAScript library, such as JQuery, that would be an imperative solution.

Web widgets are small programs embedded in a web page via HTML's *iframe* or *object* elements and they are typically implemented with HTML, CSS, and ECMAScript. There are also specific web widget environments

that contain a number of ready-made widgets from which users can choose. They also provide APIs so that developers can implement their own widgets. Examples of such web widget environments are iGoogle, Netvibes, and Pageflakes. The server push system discussed in this thesis could also be adapted for inter-widget communication on the Web. Publication VI sketches a research problem related to the communication model of inter-widget communication. Although the article presents a rough model, several open issues remain to be solved before the model can be applied.

Bibliography

[1] Murray Altheim and Shane McCarron (editors). 2001. XHTML™ 1.1 - Module-based XHTML. W3C Recommendation. URL http://www.w3.org/TR/xhtml11/.

[2] Michele Angelaccio and Berta Buttarazzi. 2006. A Performance Evaluation of Asynchronous Web Interfaces for Collaborative Web Services. In: Frontiers of High Performance Computing and Networking – ISPA 2006 Workshops, volume 4331/2006 of *Lecture Notes in Computer Science*, pages 864–872. Springer. ISBN 978-3-540-49860-5.

[3] Vidur Apparao et al. (editors). 1998. Document Object Model (DOM) Level 1 Specifications - Version 1.0. W3C Recommendation. URL http://www.w3.org/TR/REC-DOM-Level-1/.

[4] Doris Appleby and Julius J. Vandekopple. 1997. Programming Languages - Paradigm and Practice. WCB/McGraw-Hill.

[5] Jonny Axelsson, Chris Cross, Jim Ferrans, Gerald McCobb, T. V. Raman, and Les Wilson (editors). 2004. XHTML+Voice Profile 1.2. VoiceXML Forum.

[6] Robin Berjon (editor). 2006. Remote Events for XML (REX) 1.0. W3C Working Draft.

[7] Tim Berners-Lee, Roy Thomas Fielding, and Larry Masinter (editors). 2005. Uniform Resource Identifier (URI): Generic Syntax. IETF. URL http://tools.ietf.org/rfc/rfc3986.txt.

[8] Paul V. Biron and Ashok Malhotra (editors). 2004. XML Schema Part 2: Datatypes Second Edition. W3C Recommendation. URL http://www.w3.org/TR/xmlschema-2/.

[9] Peter Bojanic. 2003. The Joy of XUL. Mozilla Developer Center. URL http://www.mozilla.org/projects/xul/joy-of-xul.html.

[10] Bert Bos et al. (editors). 2009. Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification. W3C Candidate Recommendation. URL http://www.w3.org/TR/CSS21/.

[11] John M. Boyer (editor). 2007. XForms 1.1. W3C Candidate Recommendation. URL http://www.w3.org/TR/xforms11/.

[12] Engin Bozdag, Ali Mesbah, and Arie van Deursen. 2008. Performance Testing of Data Delivery Techniques for AJAX Applications. Technical report, Delft University of Technology, Delft, Netherlands. URL

`http://swerl.tudelft.nl/twiki/pub/Main/TechnicalReports/TUD-SERG-2008-009.pdf`.

[13] Tim Bray et al. (editors). 2008. Extensible Markup Language (XML) 1.0 (Fifth Edition). W3C Recommendation. URL `http://www.w3.org/TR/REC-xml/`.

[14] Dick C.A. Bulterman et al. (editors). 2008. Synchronized Multimedia Integration Language (SMIL 3.0). W3C Recommendation. URL `http://www.w3.org/TR/SMIL3/`.

[15] Richard Cardone, Danny Soroker, and Alpana Tiwari. 2005. Using XForms to simplify Web programming. In: WWW '05: Proceedings of the 14th international conference on World Wide Web, pages 215–224. ACM Press, New York, NY, USA. ISBN 1-59593-046-9.

[16] James Clark and Steve DeRose (editors). 1999. XML Path Language (XPath). W3C Recommendation. URL `http://www.w3.org/TR/xpath/`.

[17] Philip R. Cohen. 1992. The role of natural language in a multimodal interface. In: UIST '92: Proceedings of the 5th annual ACM symposium on User interface software and technology, pages 143–149. ACM Press, New York, NY, USA. ISBN 0-89791-549-6.

[18] SQLite Consortium. SQLite. Web Page `http://www.sqlite.org/`.

[19] Pavan Deolasee, Amol Katkar, Ankur Panchbudhe, Krithi Ramamritham, and Prashant Shenoy. 2001. Adaptive push-pull: disseminating dynamic web data. In: WWW '01: Proceedings of the 10th international conference on World Wide Web, pages 265–274. ACM, New York, NY, USA. ISBN 1-58113-348-0.

[20] Dojo Foundation. Dojo Toolkit. Web Page, `http://dojotoolkit.org/`.

[21] Micah Dubinko, Leigh L. Klotz, Roland Merrick, and T. V. Raman (editors). 2003. XForms 1.0. W3C Recommendation. URL `http://www.w3.org/TR/2003/REC-xforms-20031014/`.

[22] ECMA. 2009. ECMAScript Language Specification, 5th Edition. Standard, ECMA International.

[23] ECMA. 2010. ES Wiki. URL `http://wiki.ecmascript.org/doku.php`.

[24] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. 2003. The Many Faces of Publish/Subscribe. ACM Computing Surveys 35, no. 2, pages 114–131.

[25] Roy Thomas Fielding et al. (editors). 1999. Hypertext Transfer Protocol – HTTP/1.1. IETF. URL `http://www.ietf.org/rfc/rfc2616.txt`.

[26] Anthony C.W. Finkelstein, Andrea Savigni, Werner Retschitzegger, Gerti Kappel, Wieland Schwinger, and Christian Feichtner. 2002. Ubiquitous Web Application Development - A Framework for Understanding. In: Proc. of SCI2002, pages 431–438.

[27] James Foley, Won Chul, Srdjan Kovacevic, and Kevin Murray. 1988. The User Interface Design Environment. SIGCHI Bull. 20, no. 1, pages 77–78.

[28] José Manuel Cantera Fonseca (editor). 2010. Model-Based UI XG Final Report. W3C Incubator Group Report. URL http://www.w3.org/2005/Incubator/model-based-ui/XGR-mbui/.

[29] N. Freed and N Borenstein. 1996. Multipurpose Internet Mail Extensions. RFC 2045, IETF. URL http://www.ietf.org/rfc/rfc2045.txt.

[30] Jesse James Garrett. 2005. Ajax: A New Approach to Web Applications. Web Article. URL http://adaptivepath.com/publications/essays/archives/000385.php.

[31] Franca Garzotto. 2001. Ubiquitous Web Applications. Advances in Databases and Information Systems 2151, page 1.

[32] A. Ginige and S. Murugesan. 2001. Web Engineering: An Introduction. Multimedia, IEEE 8, no. 1, pages 14 –18.

[33] Ben Goodger, Ian Hickson, David Hyatt, and Chris Waterson. 2001. XML User Interface Language (XUL) 1.0. Mozilla. URL http://www.mozilla.org/projects/xul/xul.html.

[34] Google. Google Web Toolkit. Web Page http://code.google.com/webtoolkit/.

[35] Alexander Gruenstein, Ian McGraw, and Ibrahim Badr. 2008. The WAMI toolkit for developing, deploying, and evaluating web-accessible multimodal interfaces. In: Proceedings of the 10th international conference on Multimodal interfaces, ICMI '08, pages 141–148. ACM, New York, NY, USA. ISBN 978-1-60558-198-9. URL http://doi.acm.org/10.1145/1452392.1452420.

[36] Manfred Hauswirth and Mehdi Jazayeri. 1999. A Component and Communication Model for Push Systems. SIGSOFT Softw. Eng. Notes 24, no. 6, pages 20–38.

[37] G.J. Heijenk, Xinli Hou, and I.G. Niemegeers. 1994. Communication systems supporting multimedia multi-user applications. Network, IEEE 8, no. 1, pages 34 –44.

[38] Ian Hickson (editor). 2009. Web Storage. W3C Working Draft. URL http://www.w3.org/TR/2009/WD-webstorage-20091222/.

[39] Ian Hickson (editor). 2010. Web SQL Database. W3C Working Draft. URL http://www.w3.org/TR/2009/WD-webdatabase-20091222/.

[40] Ian Hickson and David Hyatt (editors). 2010. HTML5. W3C Working Draft. URL http://www.w3.org/TR/2010/WD-html5-20100304/.

[41] M. Honkala and P. Vuorimaa. 2004. A Configurable XForms Implementation. In: Proceedings of the IEEE Sixth International Symposium on Multimedia Software Engineering (ISMSE'04). IEEE.

[42] Mikko Honkala, Oskari Koskimies, and Markku Laine. 2007. Connecting XForms to Databases - An Extension to the XForms Markup Language. Position paper, W3C Workshop on Ubiquitous Web Applications. URL http://www.w3.org/2007/02/dmdwa-ws/Papers/oskari-koskimies.pdf.

[43] Yongqiang Huang and Hector Garcia-Molina. 2004. Publish/Subscribe in a Mobile Environment. Wirel. Netw. 10, no. 6, pages 643–652.

[44] Daniel Ingalls, Krzysztof Palacz, Stephen Uhler, Antero Taivalsaari, and Tommi Mikkonen. 2008. The Lively Kernel A Self-supporting System on a Web Page , volume 5146/2008 of *Lecture Notes in Computer Science*, chapter 2, pages 31–50. Springer, Berlin / Heidelberg.

[45] J., Myers B., Harris T.K., Rosenfeld R., Shriver S., Higgins M., and Hughes J. Nichols. 2002. Requirements for Automatically Generating Multi-Modal Interfaces for Complex Appliances. In: ICMI '02: Proceedings of the 4th IEEE International Conference on Multimodal Interfaces, page 377. IEEE Computer Society, Washington, DC, USA. ISBN 0-7695-1834-6.

[46] Ian Jacobs and Norman Walsh (editors). 2004. Architecture of the World Wide Web, Volume One. W3C Recommendation. URL http://www.w3.org/TR/webarch/.

[47] Jack Jansen and Dick C.A. Bulterman. 2008. Enabling adaptive time-based web applications with SMIL state. In: DocEng '08: Proceeding of the eighth ACM symposium on Document engineering, pages 18–27. ACM, New York, NY, USA. ISBN 978-1-60558-081-4.

[48] Mehdi Jazayeri. 2007. Some Trends in Web Application Development. In: FOSE '07: 2007 Future of Software Engineering, pages 199–213. IEEE Computer Society, Washington, DC, USA. ISBN 0-7695-2829-5.

[49] Gerti Kappel, Birgit Proll, Werner Retschitzegger, and Wieland Schwinger. 2003. Customisation for ubiquitous web applications: a comparison of approaches. Int. J. Web Eng. Technol. 1, no. 1, pages 79–111.

[50] Laszlo Systems, Inc. 2008. OpenLaszlo Application Developer's Guide. Technical report, Laszlo Systems, Inc. Available online: http://www.openlaszlo.org/lps4/docs/developers/.

[51] J. W. Lloyd. 1995. Declarative Programming in Escher. Technical report, University of Bristol, UK. URL http://www.cs.bris.ac.uk/Publications/Papers/1000073.pdf.

[52] Stephane H. Maes and Vijay Saraswat (editors). 2003. Multimodal Interaction Requirements. W3C NOTE. URL http://www.w3.org/TR/2003/NOTE-mmi-reqs-20030108/.

[53] G. L. Martin. 1989. The utility of speech input in user-computer interfaces. Int. J. Man-Mach. Stud. 30, no. 4, pages 355–375.

[54] Shane McCarron, Steven Pemberton, and T.V. Raman (editors). 2003. XML Events. W3C Recommendation. URL http://www.w3.org/TR/xml-events/.

[55] Scott McGlashan et al. (editors). 2004. Voice Extensible Markup Language (VoiceXML) Version 2.0. W3C Recommendation. URL http://www.w3.org/TR/voicexml20/.

[56] Nikunj Mehta (editor). 2010. Indexed Database API. W3C Working Draft. URL http://www.w3.org/TR/2010/WD-IndexedDB-20100105/.

[57] Microsoft. JScript (Windows Script Technologies). Web Page, http://msdn.microsoft.com/en-us/library/hbxc2t98.aspx.

[58] Microsoft. Windows Presentation Foundation. Web Page http://msdn.microsoft.com/en-us/library/ms754130.aspx.

[59] Tommi Mikkonen and Antero Taivalsaari. 2007. Web Applications - Spaghetti Code for the 21st Century. Technical Report TR-2007-166, Sun Microsystems. URL http://research.sun.com/techrep/2007/abstract-166.html.

[60] Peter Millard, Peter Saint-Andre, and Ralph Meijer. 2008. XEP-0060: Publish-Subscribe. XMPP Standards Foundation. URL http://xmpp.org/extensions/xep-0060.html.

[61] Izidor Mlakar and Matej Rojc. 2009. Platform for flexible integration of multimodal technologies into web application domain. In: Proceedings of the 8th WSEAS International Conference on E-Activities and information security and privacy, E-ACTIVITIES'09/ISP'09, pages 116–121. World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, USA. ISBN 978-960-474-143-4. URL http://portal.acm.org/citation.cfm?id=1736186.1736207.

[62] Mozilla. XBL 1.0 Reference. Web Page https://developer.mozilla.org/en/XBL/XBL_1.0_Reference.

[63] Mozilla Developer Center. JavaScript. Web Page, https://developer.mozilla.org/en/JavaScript.

[64] Myers, Brad A. and Rosson, Mary Beth. 1992. Survey on user interface programming. In: CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems, pages 195–202. ACM, New York, NY, USA. ISBN 0-89791-513-5.

[65] OpenLaszlo. OpenLaszlo Reference Guide. Web Page http://www.openlaszlo.org/lps4.2/docs/reference/.

[66] Ian Paterson, Dave Smith, and Peter Saint-Andre. 2007. Bidirectional-streams Over Synchronous HTTP (BOSH). Standards Track 0124, XMPP Standards Foundation.

[67] John F. Patterson, Ralph D. Hill, Steven L. Rohall, and Scott W. Meeks. 1990. Rendezvous: an architecture for synchronous multi-user applications. In: CSCW '90: Proceedings of the 1990 ACM conference on Computer-supported cooperative work, pages 317–328. ACM, New York, NY, USA. ISBN 0-89791-402-3.

[68] Robert Peacock. 2000. Distributed Architecture Technologies. IT Professional 2, pages 58–60.

[69] Steven Pemberton et al. (editors). 2000. XHTML 1.0: The Extensible HyperText Markup Language. W3C Recommendation. URL http://www.w3.org/TR/2000/REC-xhtml1-20000126/.

[70] K. Pihkala, M. Honkala, and P. Vuorimaa. 2002. A Browser Framework for Hybrid XML Documents. In: Internet and Multimedia Systems and Applications, IMSA 2002. IMSA.

[71] Mikko Pohja, Mikko Honkala, Miemo Penttinen, Petri Vuorimaa, and Panu Ervamaa. 2007. Web User Interaction – Comparison of Declarative Approaches. In: Web Information Systems and Technologies, volume 1 of *Lecture Notes in Business Information Processing*, pages 190–203. Springer Berlin Heidelberg. ISBN 978-3-540-74062-9. URL http://lib.tkk.fi/Diss/2007/isbn9789512285662/article8.pdf.

[72] Prototype Core Team. Prototype Javascript Framework. Web Page http://www.prototypejs.org/.

[73] Angel R. Puerta and Pedro Szkeley. 1994. Model-Based Interface Development. In: CHI '94: Conference companion on Human factors in computing systems, pages 389–390. ACM, New York, NY, USA. ISBN 0-89791-651-4.

[74] Dave Raggett, Jenny Lam, Ian Alexander, and Michael Kmiec. 1998. Raggett on HTML 4, chapter 2. Addison Wesley.

[75] Dave Raggett, Arnaud Le Hors, and Ian Jacobs (editors). 1999. HTML 4.01. W3C Recommendation. URL http://www.w3.org/TR/html401/.

[76] Ariel Ortiz Ramirez. 2000. Three-Tier Architecture. Linux Journal URL http://www.linuxjournal.com/article/3508.

[77] John Resig. 2006. Pro JavaScript[TM]Techniques, chapter 14, pages 287–304. Springer-Verlag, New York, NY, USA.

[78] Alex Russell. 2006. Comet: Low Latency Data for the Browser. Weblog. URL http://alex.dojotoolkit.org/?p=545.

[79] Peter Saint-Andre (editor). 2004. Extensible Messaging and Presence Protocol (XMPP): Core. IETF. URL http://www.ietf.org/rfc/rfc3920.txt.

[80] Peter Saint-Andre. 2005. Streaming XML with Jabber/XMPP. IEEE Internet Computing 9, no. 5, pages 82–89.

[81] Michail Salampasis, Christos Kouroupetroglou, and Athanasios Manitsaris. 2005. Semantically Enhanced Browsing for Blind People in the WWW. In: HYPERTEXT '05: Proceedings of the sixteenth ACM conference on Hypertext and hypermedia, pages 32–34. ACM, New York, NY, USA. ISBN 1-59593-168-6.

[82] Patrick Schmitz. 2001. The SMIL 2.0 Timing and Synchronization model. Technical Report MSR-TR-2001-01, Microsoft Research, Redmond, WA 98052, USA. URL http://research.microsoft.com/apps/pubs/default.aspx?id=69839.

[83] Orit Shaer et al. (editors). 2009. ACM Transactions on Computer-Human Interaction (TOCHI), volume 16, chapter 4. ACM, New York, NY, USA.

[84] Tony C. Shan and Winnie W. Hua. 2006. Taxonomy of Java Web Application Frameworks. In: ICEBE '06: Proceedings of the IEEE International Conference on e-Business Engineering, pages 378–385. IEEE Computer Society, Washington, DC, USA. ISBN 0-7695-2645-4.

[85] B Shneiderman. 1992. Designing the user interface: strategies for effective human-computer interaction. Addison-Wesley, 2nd edition.

[86] Ben Shneiderman. 1983. Direct Manipulation: A Step Beyond Programming Languages. Computer 16, no. 8, pages 57–69.

[87] Rainer Simon, Michael Jank Kapsch, and Florian Wegscheider. 2004. 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. ACM Press, New York, NY, USA. ISBN 1-58113-912-8.

[88] David F. Sklar and Andy van Dam. 2005. An Introduction to Windows Presentation Foundation. Windows Vista Technical Articles, Microsoft Developer Network (MSDN). URL `http://msdn2.microsoft.com/en-us/library/aa480192.aspx`.

[89] The jQuery Project. jQuery. Web Page `http://jquery.org/`.

[90] S. Trewin, G. Zimmermann, and G. Vanderheiden. 2004. Abstract Representations as a Basis for Usable User Interfaces. Interacting with Computers 16, no. 3, pages 477–506.

[91] Anne van Kesteren (editor). 2008. HTML 5 differences from HTML 4. W3C Working Draft. URL `http://www.w3.org/TR/html5-diff/`.

[92] Anne van Kesteren (editor). 2008. The XMLHttpRequest Object. W3C Working Draft. URL `http://www.w3.org/TR/2008/WD-XMLHttpRequest-20080415/`.

[93] Iris Vessey, V. Ramesh, and Robert L. Glass. 2005. A unified classification system for research in the computing disciplines. Information and Software Technology 47, no. 4, pages 245 – 255. URL `http://www.sciencedirect.com/science/article/B6V0B-4DFT6VK-5/2/5 c473276eb94e2d2e4e819c3610a8943`.

[94] Petri Vuorimaa, Dick C.A. Bulterman, and Pablo Cesar (editors). 2008. SMIL Timesheets 1.0. W3C Working Draft. URL `http://www.w3.org/TR/timesheets/`.

[95] Petri Vuorimaa, Teemu Ropponen, Niklas von Knorring, and Mikko Honkala. 2002. A Java based XML browser for consumer devices. In: 17th ACM Symposium on Applied Computing, pages 1094–1099. Madrid, Spain.

[96] W3C. Document Object Model FAQ. Web Page `http://www.w3.org/DOM/faq.html`.

[97] W3C. Report of the REX PAG 2007 on REX 1.0. Web Page `http://www.w3.org/2006/rex-pag/rex-pag-report.html`.

[98] Kuansan Wang. 2002. SALT: A Spoken Language Interface For Web-Based Multimodal Dialog Systems. In: Proceedings of the 7th International Conference on Spoken Language Processing (ICSLP'02). URL `http://research.microsoft.com/srg/papers/2002-kuansan-icslp.pdf`.

[99] Chris Waterson. 2004. Notes on HTML Reflow. Technical report, Mozilla. URL `http://www.mozilla.org/newlayout/doc/reflow.html`.

[100] Weiquan Zhao and David Kearney. 2003. Deriving Architectures of Web-Based Applications. In: Web Technologies and Applications, volume 2642/2003 of *Lecture Notes in Computer Science*. Springer. ISBN 978-3-540-02354-8.

Errata

Publication VI

In Section 4.2, it is said that the communication paradigm is based on Push-and-Pull algorithm defined by Deolasee et al. [19]. To be exact, the presented paradigm uses both a push and a pull protocols, but does not implement the algorithm in question.

The World Wide Web has expanded from a huge information storage repository into a worldwide application platform. Despite all the benefits of the Web, web applications are suffering because they are developed using the same technologies as the static documents on the Web. Additionally, new usage contexts have brought with them new requirements for web applications, which are no longer used only via Graphical User Interfaces. Recently, several parties have developed specialized technologies for web application development. The goal of this thesis is to analyze those novel web technologies and propose improvements to the technologies and architecture where applicable. The proposed improvements relate to multimodal interaction, server push, and remote UI updates especially on the developers' point-of-view. This thesis also discusses software that supports the improvements and XML-based web technologies.

BUSINESS +
ECONOMY

ART +
DESIGN +
ARCHITECTURE

SCIENCE +
TECHNOLOGY

CROSSOVER

DOCTORAL
DISSERTATIONS