Helsinki University of Technology
Publications in Telecommunications Software and Multimedia
Teknillisen korkeakoulun tietoliikenneohjelmistojen ja multimedian julkaisuja
Espoo 2006 TML-A16

WEB USER INTERACTION - A DECLARATIVE APPROACH BASED ON XFORMS

Mikko Honkala

Dissertation for the degree of Doctor of Science in Technology to be presented with due permission of the Department of Computer Science and Engineering, for public examination and debate in Auditorium T2 at Helsinki University of Technology (Espoo, Finland) on the 12th of January, 2007, at 12 noon.

Helsinki University of Technology
Department of Computer Science and Engineering
Telecommunications Software and Multimedia Laboratory

Teknillinen korkeakoulu
Tietotekniikan osasto
Tietoliikenneohjelmistojen ja multimedian laboratorio

Distribution:
Helsinki University of Technology
Telecommunications Software and Multimedia Laboratory
P.O.Box 5400
FIN-02015 HUT
Tel. +358-9-451 2870
Fax. +358-9-451 5014

ISBN-13 978-951-22-8565-5
ISBN-10 951-22-8565-7
ISSN 1456-7911

ISBN-13 978-951-22-8566-2 (PDF)
ISBN-10 951-22-8566-5 (PDF)
ISSN 1455 9722 (PDF)

URL: http://lib.tkk.fi/Diss/

Otamedia Oy
Espoo 2006

ABSTRACT

Author	Mikko Honkala
Title	Web User Interaction - a Declarative Approach Based on XForms
Published	Doctoral thesis, Helsinki University of Technology, 2006
Keywords	XML, User Interfaces, User Interaction, XForms, UIDL, XHTML

This thesis studies next-generation web user interaction definition languages, as well as browser software architectures. The motivation comes from new end-user requirements for web applications: demand for higher interaction, adaptation for mobile and multimodal usage, and rich multimedia content. At the same time, there is a requirement for non-programmers to be able to author, customize, and maintain web user interfaces.

Current user interface tools do not support well these new kinds of requirements. Thus, the main research problem of this Thesis is the definition of a device and modality independent model for high-interaction web user interfaces.

This Thesis proposes a set of criteria for user interface tools, and evaluates current tools against the criteria. It proposes a taxonomy of tools based on authoring style, consisting of procedural, declarative and hybrid tools. Based on an analysis, declarative languages are chosen, the main advantage being higher semantic level, which enables ease-of-authoring and adaptation based on context of use, current device, and user's preferences.

A layered model, consisting of mostly declarative languages, is proposed. It is composed of well-defined, and proven XML languages, and is divided into layers. The abstract UI layer contains, among others, interaction, document structure, and security, and is modality independent. The modality-dependent layer allows more detailed control over the renderings for each modality, such as visual and aural. It is shown that it is possible to automatically produce multimodal user interfaces from a single declarative user interface definition using the proposed model. This thesis focuses specifically on user interaction, where the use of XForms language is proposed. The author has co-specified the XForms language in the World Wide Web Consortium. In the proposed model, procedural scripting is only used to provide specialized modality-specific widgets in a reusable manner.

Finally, as a proof-of-concept, this thesis describes the author's implementation of the proposed model. The implementation is part of the open-source X-Smiles user agent and includes full implementations of most of the proposed languages and techniques.

TIIVISTELMÄ

Kirjoittaja	Mikko Honkala
Otsikko	Deklaratiivinen malli WWW-käyttöliittymien toteuttamiseen pohjautuen XForms-kuvauskieleen
Julkaistu	Väitöskirja, Teknillinen Korkeakoulu, 2006
Asiasanat	XML, Käyttöliittymät, Vuorovaikutus, XForms, UIDL, XHTML

Tässä väitöskirjassa tutkitaan seuraavan sukupolven WWW-käyttöliittymäkieliä ja niiden selaintoteutuksia. Tutkimuksen tärkeyttä lisää WWW-sovellusten uudet vaatimukset loppukäyttäjien taholta: suurempi interaktiivisuus, mukauttaminen kannettavaan ja multimodaaliseen käyttöön, ja multimediasisältöjen lisääntyminen. Samalla vaatimuksena on, että WWW-käyttöliittymiä voisi kehittää ja ylläpitää muutkin kuin ohjelmoijat.

Nykyiset käyttöliittymätyökalut eivät tue edellä mainittuja vaatimuksia hyvin. Siksi tämän työn päätutkimusongelma on suurempaa interaktiota vaativien WWW-sovellusten määrittelymallin suunnittelu ja toteutus. Mallin on myös oltava laite- ja modaliteettiriippumaton.

Työssä esitetään joukko kriteerejä ja arviodaan niiden perusteella tällä hetkellä käytössä olevia työkaluja. Lisäksi työssä esitetään WWW-käyttöliittymätyökalujen käyttötapaan perustuva jaottelu, jossa työkalut voidaan jakaa kolmeen luokkaan: proseduraalisiin, deklaratiivisiin ja hybridityökaluihin. Näitä analysoimalla työssä suositellaan deklaratiivisten kielten käyttöä, sillä niiden semanttinen taso on korkeampi. Tämä helpottaa niiden käyttöä, ja mahdollistaa käyttöliittymän mukauttamisen eri käyttötilanteille sopiviksi.

Työssä esitetään kerrostettu malli, joka pohjautuu kirjallisuudessa esitettyihin vaatimuksiin. Malli yhdistää aiemmin määriteltyjä ja hyviksi havaittuja XML-kieliä. Abstrakti käyttöliittymäkerros, jossa määritellä mm. interaktio, dokumentin rakenne ja tietoturva, on riippumaton modaliteetista. Alemmassa, modalitettiin sidotussa kerroksessa voidaan määritellään kullekin modaliteetille (esim. graafinen- tai puhekäyttöliittymä) sopiva esitysmuoto. Työssä keskitytään erityisesti käyttäjän vuorovaikutukseen, jonka määrittelyssä suositellaan käytettäväksi XForms-kieltä, jonka määrittelyyn tekijä on osallistunut World Wide Web Consortium:ssa.

Lopuksi työssä esitetään tekijän suunnittelema ja toteuttama prototyyppi esitetystä mallista. Se sisältyy avoimen lähdekoodin X-Smiles selaimeen.

PREFACE

This Thesis is the result of the work done at the Telecommunications Software and Multimedia Laboratory (TML), Helsinki University of Technology, between 2001 and 2006.

Financially, the work was made possible by several research projects, to whose partners I am grateful. They were funded by the National Technology Agency of Finland (TEKES), Nokia Research Center, and several other participating companies. Also, I would like to thank the Department of Computer Sciences and Engineering at Helsinki University of Technology for two grants, and Nokia Foundation for two grants, which also made life a bit easier.

In my opinion, the answer to successful research lies in the research group. Therefore I would like to express my utmost gratitude to the multimedia research group, lead by Prof. Petri Vuorimaa, who also acted as the supervisor of this thesis. He gave me trust, freedom, and support, for which I am in debt. From the group, Alessandro Cogliati, Teppo Jalava, Kari Pihkala, Mikko Pohja, and Juha Vierinen were extremely important as co-implementers of the X-Smiles browser. In addition, Dr. Pablo Cesar was the main person to engage in discussions about research methodologies, the meaning of life and everything in between. All these people also became good personal friends of mine, which I appreciate the most.

I am also grateful to the World Wide Web Consortium's XForms Working Group, which I had the privilege to be part of during the whole of my research. It taught me a lot about group work, standardization, politics, and good restaurants. Even though I cannot list all the participants here, Sebastian originally asked me to be involved and later became a good friend. The group of 2001 (Steven, TV, Roland, Leigh, Micah, John, among others) is maybe the one that I remember the best, with the later additions of the Danish vikings Kenneth, David and Allan, and the Englishmen Mark*2, among others.

Still, it cannot be denied that the main source of my well-being has been the family and friends. Therefore I would like to thank my parents and my sister Marikki for their constant support. I think you have always believed in me, even though it has not always been easy. Also, the friends from Kotka, TKK fuksiryhmä-93, and elsewhere: you really make a difference. The members of Voimaorja I want to thank for massive amounts of pure fun. Most of all, I would like to thank Annele for her love, support and trust, and for being my home.

Helsinki, December 14, 2006

Mikko Honkala

TABLE OF CONTENTS

LIST OF PUBLICATIONS

This Thesis summarizes the following articles and publications, referred to as [P1]-[P9]:

[P1] Mikko Honkala and Petri Vuorimaa. XForms in X-Smiles. *Journal of World Wide Web, Internet and Web Information Systems, 4(3), 2001, pages 151-166. Springer (formerly Kluwer).*

[P2] Kari Pihkala, Mikko Honkala, and Petri Vuorimaa. A Browser Framework for Hybrid XML Documents. *Proc. of the 6th IASTED International Conference, Internet and Multimedia Systems, and Applications, (IMSA 2002), Kauai, Hawaii, USA, August, 2002, pages 164-169. IASTED.*

[P3] John Boyer and Mikko Honkala. The XForms Computation Engine: Rationale, Theory and Implementation Experience. *Proc. of the 6th IASTED International Conference, Internet and Multimedia Systems, and Applications, (IMSA 2002), Kauai, Hawaii, USA, August, 2002, pages 196-204. IASTED.*

[P4] Mikko Pohja, Mikko Honkala, and Petri Vuorimaa. An XHTML2 Implementation. *Proc. of the Fourth International Conference on Web Engineering (ICWE2004), Munich, July, 2004, pages 402-415. Springer.*

[P5] Mikko Honkala and Petri Vuorimaa. A Configurable XForms Implementation. *Proc. of the IEEE Sixth International Symposium on Multimedia Software Engineering (MSE2004), Miami, FL, USA, December, 2004, pages 232-239. IEEE.*

[P6] Mikko Honkala, Pablo Cesar, and Petri Vuorimaa. A Device Independent XML User Agent for Multimedia Terminals. *Proc. of the IEEE Sixth International Symposium on Multimedia Software Engineering (MSE2004), Miami, FL, USA, December, 2004, pages 116-123. IEEE.*

[P7] Mikko Honkala and Petri Vuorimaa. Secure Web Forms with Client-Side Signatures. *Proc. of the Fifth International Conference on Web Engineering (ICWE2005), Sydney, Australia, July, 2005, pages 340-347. Springer.*

[P8] Mikko Pohja, Mikko Honkala, Miemo Penttinen, Petri Vuorimaa, and Panu Ervamaa. Web User Interaction - Comparison of Declarative Approaches. *Proc. of the 2nd International Conference on Web Information Systems and Technologies (WEBIST 2006), Setbal, Portugal, April, 2006, pages 295-302. Springer.*

[P9] Mikko Honkala and Mikko Pohja. Multimodal Interaction with XForms. *Proc. of the The Sixth International Conference on Web Engineering (ICWE2006), Palo Alto, California, July, 2006, pages 201-208. ACM.*

Other Related Publications

Other related publications (cf. Figure 1.1 for relationships) by the Author, which are ***not*** included as a part of this Thesis:

[RP1] Petri Vuorimaa, Teemu Ropponen, Niklas von Knorring, Mikko Honkala. A Java Based XML Browser for Consumer Devices. *Proc. of the ACM Symposium on Applied Computing, Madrid, Spain, March, 2002, pages 1094-1099. ACM.*

[RP2] Mikko Honkala and Petri Vuorimaa. Advanced UI Features in XForms. *Proc. of the 8th International Conference on Distributed Multimedia Systems, September, 2002, pages 715-722.*

[RP3] Pablo Cesar, Mikko Honkala, Vesa Kautto, Kari Pihkala, Juha Vierinen, and Petri Vuorimaa. Group Communication Enabled XML browser. *Proc. of the International Conference on Communications, Internet and Information Technology, St. Thomas, Virgin Islands, USA, November, 2002, pages 213-218. IASTED.*

[RP4] Kari Pihkala, Mikko Honkala, and Petri Vuorimaa Multimedia Web forms. *Proc. of the SMIL Europe 2003. Paris, France, February, 2003.*

[RP5] Teppo Jalava, Mikko Honkala, Mikko Pohja, and Petri Vuorimaa. Timesheets: XML timing language. *World Wide Web Consortium Member Submission, 2005. W3C.*

1 INTRODUCTION

The World Wide Web (WWW) has been a phenomenal success. Collection of simple technologies and a simple publishing model has enabled the creation of a huge information repository, which anybody with an internet connection can access. The web is changing, though. While it previously was mainly a channel for publishing and retrieving documents, it is now transforming into a platform for interactive services and applications. Even highly interactive content authoring applications are currently being implemented in the web.

This change towards more interactive applications has improved the utility and the user experience of the web, but it has also led to a rise of complexity in programming web applications. There are three main reasons for the complexity. First, generating these dynamic web pages on the fly makes programming code harder to understand and debugging more difficult. Second, the dynamic web pages are often a mixture of markup languages, client-side scripting code and server-side function calls, which makes them even harder to maintain and understand. Third, the high number of software tools in web applications makes those applications complicated to design and fragile to deploy and run. [23]

Web applications are distributed, and accessed with variety of different user terminals (e.g., mobile phones, television sets). Unfortunately, the user interface technologies, which are used today are derived from the research in the 1980's [80], where these requirements were unknown, and computing was based on single-user desktop model.

This Thesis researches software tools for web user interaction and relates to *Software Engineering* (SE) and *Human Computer Interaction* (HCI) within the Computer Science (CS) discipline. In particular, the research area is *user interface engineering*.

1.1 Current Trends in Web User Interaction

There are three somewhat separate trends, which have acted as the main motivation for this Thesis:

1. Demand for highly interactive web applications [93].
2. Demand for adaptation of web applications for mobile and multimodal usage [45].
3. Demand for multimedia content in web applications [93].

Examples of highly interactive web applications (also called Rich Web Applications [89] or Rich Internet Applications [32]) include Google Spreadsheets[1], which is a spreadsheet application that works with a web browser without any additional plugins, and Gmail[2], which

[1]Google Docs: Spreadsheets, Web Application, Available at: *http://docs.google.com/*. Accessed Nov 2006.
[2]Gmail, Web Application, Available at: *http://www.gmail.com/*. Accessed Nov 2006.

contains a Rich Text email editor, also used with a web browser. As daily tasks are done more and more using the web, the mobile and multimodal usage of web applications is becoming more important as well. The last trend, demand for multimedia content, is already being realized, as media companies are starting to provide multimedia, and TV content within their web pages, with examples such as YouTube [3], CNN Video[4].

1.2 Definitions

A *web application* is an application, whose user interface is distributed over an HTTP gateway, and which accesses a certain server-side resource (e.g., a file, a database, an information system). The user interacts with the user interface through a generic client (i.e., the browser), which interprets the UI and sends back request parameters. Request parameters are processed by the web server and passed to the web application, which dynamically generates an HTTP response [63]. The process of writing web applications is called *web development* [48].

There are many software libraries, languages, or frameworks available for defining the user interfaces of web applications. These are called *user interaction tools* in this Thesis. For instance, Java Servlets are server-side Java programs, which generate HTML markup, which is rendered in the browser.

A *device independent* user interface can be utilized in various end-user devices, while in *multimodal interaction* two or more combined input modes are processed in a coordinated manner with multimedia system output.

1.3 Problems with Current Web User Interaction Tools

The three trends defined in Section 1.1 combined with the current (legacy) web UI tools have resulted into the following main problems:

Low interactivity. Lack of higher interaction make the user experience worse compared to many desktop applications. Also, the page-centric model of the Web often breaks the user interaction at unnatural places [32].

High latency. Network overhead and unnecessary round-trips result into bad responsiveness and generally make the user experience worse [20].

Web development complexity The problem with current web development frameworks is that that even simplest changes to the pages need changes to back-end program code. So called template languages, such as Java Server Pages (JSP), make small changes easier, but they are too much HTML page-centric: the developer writes HTML pages, which then can incorporate either server-side generative code (e.g., in Java) or client-side scripting (in ECMAScript). Not surprisingly, this intermixing of server- and

[3] YouTube, Web Application, Available at: *http://www.youtube.com/*. Accessed Nov 2006.
[4] CNN Video, Web Application, Available at: *http://www.cnn.com/video/*. Accessed Nov 2006.

client-side program code along with HTML markup is difficult to write, debug, and maintain. [23].

Missing multimedia integration. Adding multimedia (video, audio) to web pages is not supported by the core technologies, and often requires third-party plugins, which are not necessarily available in all platforms [32].

Lack of device and modality independence. Even though the current web languages are somewhat platform independent, the current tools do not have good support designing and implementing device and modality independent user interfaces. The main problems are [45]: Lack of knowledge and experience, methodology, and tool support, for designing and implementing device independent user interfaces.

In fact, the current web user interaction tools were not not originally designed for today's complex application scenarios. For instance, HTML forms and procedural client-side scripting are used as the main user interaction definition, even though they were not designed to describe complex, higher-interaction UIs and to be adaptable for different contexts of use.

On the other hand, the following success factors of the web might still be desirable in the future: standards-based communication and UI definition, ubiquitous generic client, platform independence, hypertext functionality, and independence of any specific server-side technology. This Thesis focuses on declarative user interfaces description languages, which promise to be more easily adaptable to different usage contexts, while still maintaining the important features that have proven to be successful in the web.

1.4 Aim of the Study

This Thesis aims to provide user interface tools for next-generation web applications. The main goals are related to software engineering: to enable software developers and even non-programmers to build higher-interaction web applications in an efficient manner. Secondary goals are related to usability and ubiquity: by providing tools, which support adaptation, higher interaction, and multimedia content, the same development effort should lead to more usable UIs in a wide array of end-user devices and usage scenarios.

Traditionally, procedural approach, such as Toolkit APIs or scripts, have been used to provide high interactivity in user interfaces. Recent trend in the research community, as well as the industry, has been to utilize declarative languages in user interface description. The basic difference between procedural and declarative languages is that when procedural languages tell how to do things, declarative languages try to capture the idea of what to do. The main benefit of this approach is the raised semantic level, which allows user interfaces to be automatically adjusted based on usage scenario and the used device. Simon et al. call this model *single authoring* of device- and modality-independent interfaces [105]. In addition, declarative languages are easier to process by accessibility and other tools, therefore fixing many of the problems found in the approaches with lower semantic level (e.g., HTML forms and scripting).

In summary, declarative UI languages have usually a higher semantic level while traditional, procedural programming languages have more expressive power. It is essential that a balance between semantic level and expressive power is found. This balance is the focus of the research work in this Thesis.

1.5 Assumptions

In order to focus the research in this Thesis, and based on the current state of the art, the following ***assumptions*** were made.

Rich Web Applications. Users prefer web applications, which provide higher interactivity and seamlessly integrated multimedia content, compared to the ones that fail to provide this [32].

Skill set of developers. In the future, not all developers have homogeneous skill set. Low-level systems developers have more detailed knowledge and experience in operating systems, hardware, and low-level programming languages. Application back-end developers know their databases and Web services APIs. *User interface developers*, who are the target of this Thesis, have extensive, often intuitive, knowledge on usability, but are not necessarily experienced programmers. It does not make sense to offer the same tools, and expect the same skills, for each category of developers.

Heterogeneous end-user devices. New end-user device types are being invented and brought to the consumer market all the time. In order to get the widest possible audience, web applications need to be usable on many platforms [45].

XML-based languages. Even though there is a long tradition of pre-XML declarative UI languages (cf. e.g., [47, 51]), W3C defined XML languages are used in this Thesis whereever possible. W3C is the most authoritative body for web development, giving it's standards the biggest audience possible, which also means that these technologies are well-defined, widely tested and interoperable. Also, XForms [31] has been chosen as the base language for interaction, since the Author has been involved in the XForms language specification, thus being able to affect the language to fullfill central requirements.

Browser as the generic client. This Thesis assumes that the browser will be the generic client that is installed and running in the various end-user devices. The final user interfaces will be rendered the browser. There may exist different kinds of browsers, such as voice and graphical, but they will mostly support the same standardized set of input languages[5] - thus reducing the need for transcoding the UI description[6].

[5] At the time of writing, the supported set of languages in mainstream browsers, such as Internet Explorer, Opera, and Firefox, is XHTML 1.1, CSS 2.1, DOM L2, ECMAScript, and XmlHTTPRequest, and SVG is getting more and more support. Even some mobile browsers, such as the Nokia S60 3rd edition browser and Opera, have a good support of this set of technologies.

[6] There still might be other reasons for transcoding, such as removal or addition of high-level tasks based on

1.6 Research Questions

The research questions of this Thesis can be summarized as follows:

Q: A device and modality-independent model for high-interaction web user interfaces. The question can first be motivated by the end user needs to have higher interaction, device independence, and multimedia content. Second, the discipline would benefit from the model; standardization bodies could use it as input to future standards, and user agent vendors could use it to assess the implementation costs of these standards. User interface authors would benefit from a potentially more productive way of creating UIs.

This is the main research question, whose subquestions are the following:

Q1: Requirements for next-generation web UI description language. What are the requirements for next-generation web UIDL, based on the recent demands of higher interaction, device independence, and multimedia content?

Q2: Evaluation of current UI description languages. How well current UI description languages address the above requirements in Q1?

Q3: Compound document profile to fit the requirements.. How can current document formats be combined and extended so that they meet the requirements in Q1 and overcome the shortcomings of the reviewed languages in Q2? How should the layout models be integrated? Can multimodality be supported in such a profile? What are the security implications?

Q4: Implementation implications. What are the main issues when implementing the proposed document profile in Q3? Can different implementations of the profile interoperate well enough?

The research topics of this Thesis are UI Description Languages, Context Independence, User Agent Software, and Communication Protocols (cf. Figure 1.1). The research topics of this Thesis are important because of many reasons. A lot of expensive effort of the web application developers is lost in tailoring each application for different targets because the current web interaction tools do not support adaptability, usability, and maintainability well. Additionally, usability research has proven that usability has huge impact on how efficiently people can use applications.

1.7 Research Methods

This Thesis uses mainly Formulative research approach. The main research methods are Conceptual analysis[7] and Concept implementation (proof-of-concept) [52, 99].

the target platform [46].

[7] A study of the concepts presented, without using mathematical methods, for instance validating a model based on requirements. [24]

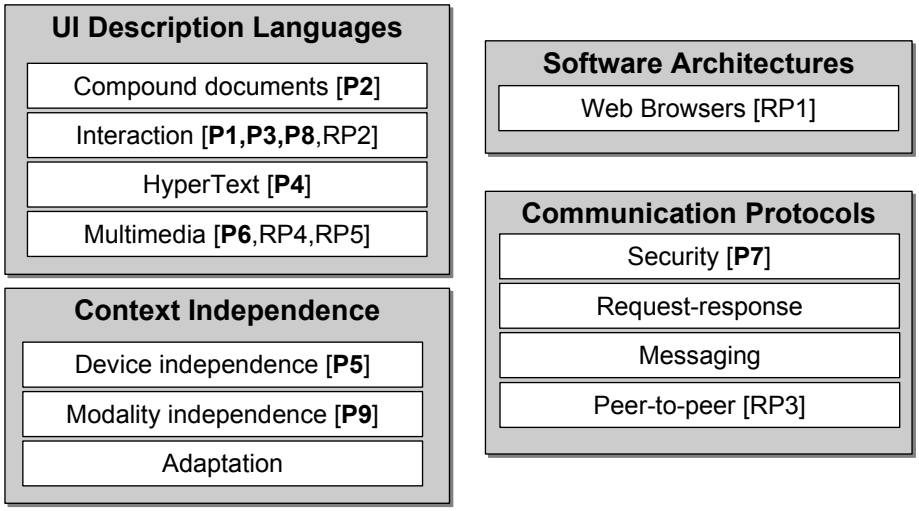

Figure 1.1: The research topic categories of this Thesis and the main category of each of the Publications. This figure also shows the relationship of the related publications to this Thesis.

The research process for the included Publications mostly follows this pattern: first, a review of literature and the researcher's own experience was used to reach a state-of-the-art knowledge on the specific topic. Next, requirements analysis was used to provide a solid base for concept implementation. The next phase was software design and implementation. Finally, use case analysis, and evaluation of the results followed. Participation in standardization process was used when applicable and possible.

The validation of this Thesis is done on two levels. First, theoretical validation is done by comparing the approach against others using general criteria and more detailed requirements, both derived from the literature. Second, empirical validation is performed by a proof-of-concept implementation, and use cases from different applications, therefore validating the applicability of the model's features and the effectiveness of its main concepts.

1.8 Organization of the Thesis

This Thesis is organized as follows: In the Section 2, the current literature is surveyed, and application scenarios are given and analyzed. It also proposes a set of evaluation criteria for web UI tools, answering to the Research Question Q1. The following Section 3 reviews the current tools based on these criteria, relating to the Research Question Q2. Then, in Section 4, the proposed model for next-generation web user interface tools is presented (Q3). Next, an implementation of the proposed model is presented in Section 5, which is used as a proof-of-concept of this study (Q4). Finally, Section 6 provides the conclusions and summary of the Author's contributions.

2 BACKGROUND

This Chapter gives the background of the research and reviews the related research on user interaction, and user interaction tools. It also derives a set of evaluation criteria for next generation user interface tools.

2.1 Definitions

This Section gives definitions, which are used throughout the Thesis (cf. Figure 2.1 for their relationships). The *user interface (UI)* of a computer program is the part that handles the output and the input to and from the person (i.e., the *user*) using the program. The rest of the program is called the *application* [78].

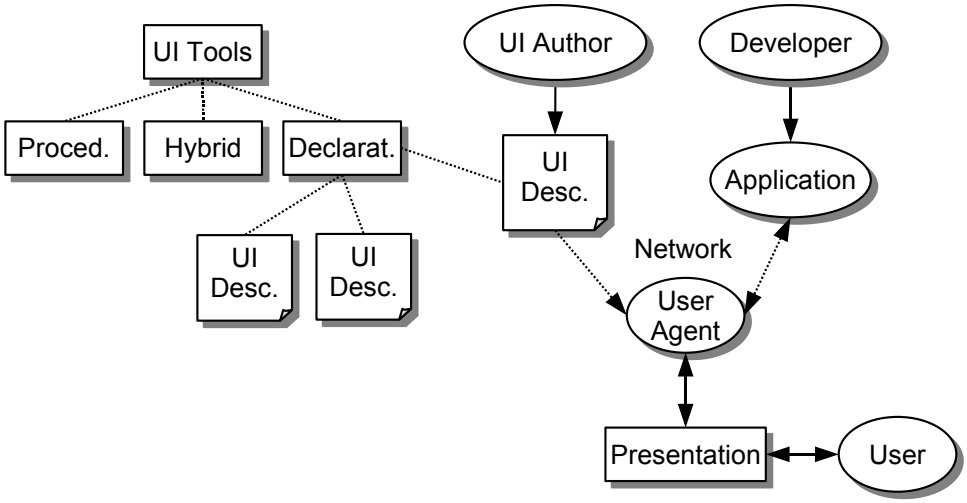

Figure 2.1: The relationship of the definitions used in this Thesis.

The people involved in developing and using user interfaces are classified as follows: The term *user* denotes the end-user, who utilizes an application through its user interface, and is not an expert on user interface design or implementation. The *UI author* is the person, who develops the user interface of an application using a user interface tool. Finally, the *developer* is a programmer, who develops the application. Sometimes the developer is also the UI author.

User interaction is the process of the user utilizing a user interface. As in [78], the general term *user interface tool* will be used for all software and libraries aimed to help create user interfaces. Note that the tool includes procedural libraries called *toolkits*, as well as higher-level declarative tools.

In this Thesis, the term *procedural UI tool* denotes a programming library, such as Abstract Windowing Toolkit, which is used within a procedural (i.e., *imperative*) language, such

as Java [24]. *Declarative UI tools* have higher semantic level than procedural tools. Typically, authoring with such a tool is possible even for non-programmers. A well-known example is the HyperText Markup Language (HTML). *User Interface Description Language (UIDL)* is another general term for declarative UI tools. A *hybrid UI tool* has properties from both declarative and procedural tool categories. A *UI description* is an entity, which describes the user interface elements, possibly along with the user interface logic.

A *user agent* is a software component (e.g., a *browser*) running in the user's device, which reads the UI description, and communicates with the application through the *network*. It generates a dynamic *presentation* of the user interface, which the user then interacts with. The interaction can happen simultaneously in different modalities, such as aural and visual.

A classic model of user interface tools is presented by Myers [78]. In that model, the topmost component is Higher-level Tools, followed by Toolkit in the middle, and Windowing System on the bottom. On top of the model lies the Application, and below, the Operating System. It is depicted in Figure 2.2, where it is extended with tool classes used in this Thesis, namely UI description instead of Application, Declarative and Hybrid UI Tools instead of Higher-level-tools and Procedural UI Tools instead of Toolkit.

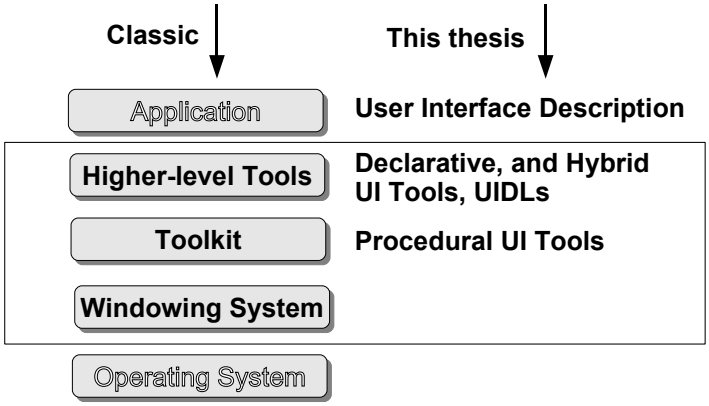

Figure 2.2: The tools discussed in this Thesis related to classic tool architecture [78].

2.2 Application Scenarios

Some examples of application scenarios for this Thesis are listed below :

Services supporting mobility. In mobile use, it is important that the web applications are device independent. In some usage contexts, such as when driving a car, they should allow multimodal usage as well, as demonstrated by the following scenario from Publication [P9]:

1. At the office. The user uses the car rental service to book a car for a holiday trip. Then she wants to know how to get to the car rental pick-up using public transportation. She uses mainly visual modality to interact with the route planner. The route planner saves the query.

2. Walking. While walking to the bus, the user remembers that they need a bigger car, so that the holiday skiing equipment would fit in. She logs in to the car rental service using her mobile phone. Using speech input, she changes the car preference, but notices that the pick-up point changes. She logs in to the route planner, and inputs the new pickup address using voice input. Finally, she uses the text message service using speech input to notify her husband that she will be picking up the rental car.

3. Driving a car. The user uses the route planner while driving, using the onboard multimodal internet browser. She interacts purely with voice to select the holiday destination (the startpoint is filled in by her GPS receiver). She uses visual output for the map of the route and aural output for the driving instructions.

Content management. This use case from Publication [P8] is a content management system of an internet magazine. The users are mainly journalists, who have experience in using typical word processing program and are familiar with concepts like copy-paste. The system allows editing simple documents, tables, and the structure of the magazine, including article release dates, etc. A slightly similar use case, albeit simpler, is found in the Publication [P4], where the structure and headings of a document, are edited.

Distance education portal. This use case is described in the Publication [P6]. The UI author of the lectures in the portal is either a teacher or an assistant. Note, that the UI author is not a programmer, so the system should be as easy to author as possible. The content is structured audio and video of the lectures, synchronized with the slides, with user controlling the flow of through the lectures (maybe skipping parts she already knows). The user of the portal is a student who wants to reach the lectures ubiquitously using different kind of devices. The UI author wants to control the learning rate of the students, so she wants to create short "exams" between or within the lectures.

Universal Remote Controller (URC). This use case is proposed by Nichols et al. (Personal universal controller PUC) [82], and by Zimmerman et al. [118, 117]. The general idea is that any user device could control any other device. For instance, the user's sauna would transmit its' user interface so that it's operation and temperature could be controlled using a mobile phone. Related work by Barton et al. [7] utilize XForms to communicate data between web servers and sensor-enabled client devices, including PDAs and wireless digital cameras. In their proposal, form data can be inputted directly with sensor data as well as with manually entered input. The web could be used as the underlying technology to transfer these user interfaces.

Workflows. Workflows often span different organizations and/or user groups, and therefore it is beneficial to have ubiquitous access to the workflow application. Since the web is an ubiquitous distribution channel, it is a natural tool to deploy Workflow applications. A newer trend is ad hoc workflows, which are more flexible than predefined workflows, and even possible for end-users to author. There are many proposals for implementing workflow applications [35, 110, 27, 114] using UIDLs or the web.

Secure transactions. Many usage scenarios, such as online banking, require secure, non-repudiable transactions. The Publication [P7] presents a secure insurance claim application. The application allows the user to file in an insurance claim, digitally signed with his PKI private key. It is possible to have multiple parties in the insurance claim. The different parties in this case should be independent of each other, so that it is possible to add and remove parties from the claim. The main requirement is non-repudiation; a signer cannot later deny having done the signature.

The above application scenarios contain common trends. For instance, many of them require *higher interaction*, e.g., interactive manipulation of structured information, and some (e.g., content management) even authoring of richly formatted structured information. Also, these applications contain ever *richer multimedia content*, including synchronized audio and video. *Usage context independence* is also a common theme; Some applications would be used in "extreme" situations, such as while driving a car. The applications should adapt to these usage contexts without additional actions from developers or users. One way of adapting is to use multimodal interaction. Some applications require *non-programmer authors* to be able to maintain the user interfaces. There is a group of people, which would be able to author or modify interactive applications, if they were given the tools. Consider spreadsheet authors, who are typically non-programmers. Finally, *security* should be a consideration, as more private data is being edited in web applications.

2.3 User Interaction Models

Research on user interface models and technologies can roughly be divided into three different categories, according to Beaudouin-Lafon [8]. On the highest abstraction level, *interaction models* model how the user interface is designed. Next, *architectural models* define more concrete architectures for realizing the user interfaces. Finally, *implementation models* are concrete libraries or languages, which allow the developer, in a certain platform, to implement a user interface. There are also *device-level models*, but they are outside of the scope of this Thesis.

Several *interaction models* (or *interaction styles*) can be found in the literature, and many of those have found their way into architectural and implementation models. Probably the most well-known interaction model is still the WIMP (Windows, Icons, Menus, and Pointing) model, originally used, e.g., in the Xerox Star user interface [106]. Direct manipulation [103] is another well-known interaction model, where the objects are manipulated directly with immediate visual feedback. Direct manipulation is used quite a lot in today's user interfaces, e.g., in drawing programs. Many post-WIMP interaction models have been proposed with varying success. For instance, Instrumental Interaction refines the direct manipulation model by generalizing and operationalizing it [8]. Direct Combination [56] is another interaction model, which is based on a metaphor of combining objects together.

Several *architecture models* can be found from the literature, as well. Maybe the most well-known is the Model-View-Controller (MVC), which separates the concerns of data,

view, and the control. [65].

Finally, *Implementation models* include all languages and libraries, which can be used to realize user interfaces on given platforms. Listing even a representative array of these, is out of the scope of this Thesis. The next Sections looks at implementation models, which are specific to the scope of this Thesis, namely web user interaction.

All of the above categories are of interest in this Thesis. For instance, the general idea of the UIDL is an architectural model, and XForms, which is a UIDL can be classified as an implementation model. On the other hand, it is important that a given architectural and implementation model supports relevant interaction models.

2.4 Taxonomy of Interactive Web Applications

Compared to traditional interactive applications, web applications have certain noticeable characteristics. Typically they are distributed, multi-user applications, with special emphasis on security, scalability, and device independence. Deshpande et al. have proposed, based on previous research [50], a taxonomy of web applications, which is based on the evolution of the web. Starting from the earliest web applications and moving towards present-day ones, their categories are [28]:

- *Informational*. Online newspapers, etc.
- *Interactive (User-provided information / Customized access)*. Registration forms, customized information presentation, games.
- *Transaction*. E-shopping, ordering goods and services, banking.
- *Workflow*. Planning and scheduling systems, inventory management, status monitoring.
- *Collaborative work environments*. Distributed authoring systems, collaborative design tools.
- *Online communities, marketplaces*. Chat groups, recommender systems, marketplaces, auctions.
- *Web Portals*. Electronic shopping malls, intermediaries.
- *Web Services*. Enterprise applications, information, and business intermediaries.

While the above taxonomy is fairly comprehensive from the web evolution point-of-view, at least until to date, it fails to classify applications based on their interactivity. Therefore, another taxonomy of interactive web applications is proposed in this Thesis. This taxonomy focuses on the complexity of user interaction.

Information retrieval Applications in this category have a low interaction requirements, and they only allow the user to browse and search information in some database or set of documents. The amount of user interaction is limited to entering simple search terms and navigating links. A typical web search engine is an example of information retrieval application. Also, navigating a web site with semantic linking would fall under this category. From [28] the *Informational*, and *Web Portals*, classes falls under this category.

Information manipulation This class of applications has medium interaction requirements, since only certain, predefined, pieces of information is edited by the user. An analogy to desktop applications would be using a ready-made spreadsheet application, e.g., for travel expenses claim, where modifications are allowed only in predefined cells. Another example is adding and removing items to a shopping cart in a e-commerce application. From [28] the *Interactive, Transactions*, and *Online communities* classes can be included in this category.

Information authoring requires a much greater level of interactivity. Applications in this class require application-specific interactors, and even interaction styles, such as direct manipulation. Word processor, spreadsheet, or image drawing applications are examples of information authoring. From [28] the *Collaborative work environments*, and *Workflow* classes fall under this category.

Note that from [28], the only class that is not included in the new taxonomy is *Web Services*, which do not usually have user interaction themselves, but rather are used in communication between computers.

2.5 Adaptation

UI *adaptation* is an important means to achieve device independence. It can be defined as the ability of the UI to run and be usable in different usage contexts and devices. The main reason to do UI adaptation arises from different devices having various output and input methods (e.g., smaller or bigger screens, voice output, mouse, or keyboard) and the users having various usage contexts.

Adaptation methods can be divided based on the *adaptation target* into the following categories [91]:

Spatial adaptation. The spatial arrangement of the objects is altered.

Temporal adaptation. The timeline of the presentation or application is altered.

Interaction adaptation. The interaction is adapted, e.g., by replacing interactors for other interactors more suitable in the context.

Media adaptation. Media format or quality is altered.

Several adaptation techniques have been proposed in the literature. Most of the proposals can be categorized into one of the following categories based on Florins, who has conducted an extensive evaluation of these methods [45]:

1. **Multiple authoring**, which is the traditional development approach, where the UI of each device is authored separately.

2. **Unique portable code**, which consists of authoring a single UI, which is able to run on different devices. This can be further divided into a) *virtual toolkits,* and b) *generic clients* (e.g., a web browser for markup languages) [45]. The latter method can be extended with author defined rules or hints [54] for adaptation.

3. **Transcoding** takes as input a UI's lower-level code, and generates UI codes for other devices.

4. **Multireification** takes as input a UI's high-level specification, and generates other UI descriptions for other devices.

5. **Abstraction-reification** takes as input an lower-level UI description, generates an intermediate, higher abstraction level representation and produces the other UI descriptions.

Items 2-5 often promote a property called *Single-authoring*, which can be defined as "centring the design process on a source interface designed for the least constrained platform" [45]. Sometimes this term is used more strictly to mean "a single, device-independent user interface description, which can be mapped to a good concrete UI for each feasible target device"[1].

An interesting contribution for classifying and evaluating different technologies for UI adaptation is the Cameleon reference framework [22]. It claims that a typical model-based approach divides the UI description to different layers of abstraction. These layers are, starting from the highest level of abstraction:

1. **Task & Concepts** describes the various tasks to be carried out and the concepts required by these tasks.

2. **Abstract UI** defines presentation units by grouping subtasks according to various criteria. It also contains a navigation scheme between the presentation. An abstract UI is independent of any modality of interaction.

3. **Concrete UI** concretes an abstract UI for a given context of use to define widgets layout and interface navigation. It is still independent of any computing platform.

[1]Braun, E., Hartl, A., Kangasharju, J. & Muhlhauser, M. (2004). Single Authoring for Multi-Device Interfaces. Proceedings of the 8th ERCIM Workshop "User Interfaces For All" (28-29 June, Vienna). Retrieved Nov, 2006, from http://ui4all.ics.forth.gr/workshop2004/publications/adjunct-proceedings.html

4. **Final UI** is the operational UI i.e., any UI running on a particular platform either by interpretation (e.g., through a web browser) or by execution (e.g., after compilation of code in an interactive development environment).

Figure 2.3 shows the different operations, which are possible between the layers. First, *abstraction* is an operation, which takes a lower abstraction level description and transforms it to higher abstraction level description. Second, *reification* is the opposite operation to abstraction. The third operation is *translation*, which maintains the abstraction level, but takes into account the context of use.

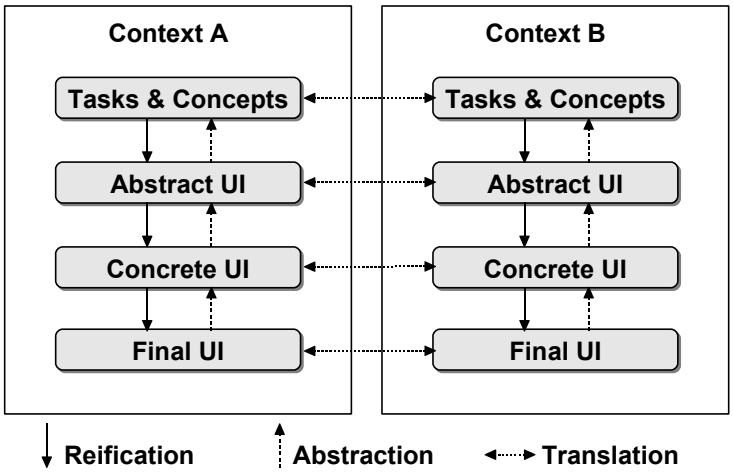

Figure 2.3: The different levels of abstraction and the operations between them and different contexts of use [22].

2.6 General Evaluation Criteria

In order to evaluate different approaches to user interface generation, a set of general evaluation criteria needs to be defined. In previous research, different sets of criteria has been proposed. The following list contains criteria based on the scope of this Thesis.

Myers has defined a good set of general criteria for user interaction tools [78]. Some UIDL specific criteria can also be found in other papers [108, 112]. The following criteria is extended from those papers.

Expressivity How much can be expressed with the tool, considering the final concrete user interface. For instance, does it limit the interaction styles [108]. This basic criterion does not take into account adaptation for different devices. Myers calls this *Ceiling* [80]. This criterion can be further divided based on the usage scenarios in Section 2.2 to expressivity in a) *Visual Presentation*, b) *Interaction*, and c) *Multimedia and synchronization*.

Ease of learning How easy is it to start using the tool. This criterion depends on the target authors, and should finally be empirically proven. For instance, are authors only

limited to programmers, or can they be graphical designers, or even end users? One can argue that this criterion is influenced by how well the tool matches the developers mental model of the task at hand[2]. Myers has a similar criterion, called *Threshold* [80].

Ease of authoring How easy is it to author UIs with the tool. How much effort does it take to maintain UIs based on this tool. This is called *Ease-of-use of tools* in [78].

Extensibility Can the tool be extended with new interactors and interaction styles, or is it limited to the currently available ones? Also called openness [108].

Authoring tool interoperability Some approaches make it almost impossible to create interoperable authoring tools. For instance, procedural programming code (e.g., ECMAScript, Java, or C++), generated by a graphical authoring tool, is impossible to read into another tool, because of lack of higher-level semantics [102]. This is often circumvented by using higher-level descriptions, such as in Gnome Glade[3].

2.7 Web-specific Evaluation Criteria

The above general criteria can be applied to any user interaction tool, but since this Thesis is focused on web user interaction, the following set of web-specific criteria is defined:

Latency Latency is the perceived slowness of the user interface. Latency in web applications happens at multiple levels. First, when a new page is requested from the server, there is a latency, which depends on many factors (e.g., on network connectivity, size of files, and number of simultaneous users) [72]. Second, latency within a web-page depends on the UI description, UIDL, and the user agent. As a general rule, the longer the latency, the worse the user experience [10]. Myers calls this criterion *performance of resulting interface* [78], although in here the network latency plays a bigger role.

Device independence Advances in hardware has enabled the building of efficient and cheap computers that are disguised in many forms, such as mobile phones or car navigation systems. Similarly, advances in wireless and wired communications technology have permitted IP-based communication between those devices. These trends have created a need to provide device-independent access to the services. *Adaptation* is a typical means to achieve device independence. A subcriterion is *predictability*, which measures whether or not there are unpredictable changes in the user interface in different devices. This is also called Portability [78].

Modality independence means that the user interface can be used in different input modalities and output medias (such as visual and aural), and even if the user interface was not originally designed for those modalities.

[2]This was pointed out to me by Dr. Pablo Cesar in a late night discussion in June 2006.
[3]Glade. UIDL and software library. Available at: *http://glade.gnome.org/*. Accessed Nov 2006.

Security The biggest difference between desktop and web applications is that web applications are *distributed* by nature. There are many architectural styles for distributing applications [44], but security threats are increasingly common, especially if financial, personal, or otherwise confidential information is being transmitted. Another important property is non-repudiability, which is often required in these usage scenarios.

Accessibility Several countries have enacted legislation, which require public web applications to be accessible to people with disabilities [59][4].

Web integration In order for a new technology to be successful in the web, it needs to integrate to the underlying technologies and protocols. This needs to happen at many levels: addressing (URL), transport (HTTP), syntax (XML), events (DOM), and styling and layout (CSS). For instance, even though Java is superior to ECMAScript in many aspects, the latter has become dominant as a web user interface tool, since it integrates better to the web model [44]. Of course, for a good reason, a technology can deviate from these.

Ease of implementation Since the web is used by browsers running in various end-user devices, it is important that the languages are as easy as possible to implement. For instance, dynamic vector graphics support in browsers (SVG) is still in its infancy, since the implementation is rather difficult.

[4]W3C WAI Policies, Available at *http://www.w3.org/WAI/Policy/*. Accessed Nov 2006.

3 EVALUATION OF USER INTERFACE TOOLS

This chapter evaluates the most relevant UI tools for web user interaction based on the criteria developed in the previous section.

3.1 Procedural UI Tools

The research in this Thesis is in part based on the research on user interface tools, which dates back many decades. Many research systems in the 1960's, such as NLS [36], had the idea of displaying multiple windows at the same time. Overlapping windows were proposed by Alan Kay in 1969 in his PhD Thesis [61]. Later he used them in his Smalltalk system. Many commercial systems picked up the idea, and now a windowing system is the mostly used user interface model. The major operating systems, MacOS, Linux, and Windows, all have their own windowing system, but all of them offer similar services for the application developer, and similar user experience for the user.

A windowing system can be divided into three basic components, *Windowing system, Toolkit, and Higher level Tools* (cf. Figure 2.2). *The windowing system* (e.g., X Windows) controls and monitors graphical contexts and separates them into different, possibly overlapping, regions (i.e., windows). *The toolkit* is a set of widgets (e.g., buttons, menus, etc) that can be used by application programs. The main advantage is that using the standard widgets provided by the toolkit, the user experience is equal between different applications. The toolkit usually also provides layout functionality, therefore simplifying the implementation of applications for varying sizes of windows and displays. Finally, *Higher-Level tools* come in different flavors, some examples include event, declarative, and constraint languages [79].

The user can interact at different levels of the system. For instance, she can change the size and position of a window (Windowing system level) or click a button (Toolkit level). The toolkit has the following responsibilities [24]:

- **Interaction:** to handle user input.
- **Canvas Operations:** the actual rendering region, canvas [83], and graphics primitives such as rectangle or triangle.
- **Set of Widgets:** predefined user interface elements (e.g., Button, Text Field).
- **Graphical Layout:** to control the location of the widgets.

There is one procedural approach, which is in widespread use in the web, namely Java Applets.

Java Applets

The most well known web UI framework, which uses the toolkit approach, is *Java Applets*. Java Applets allow a secure way of deploying complex application UIs, which are written in the object oriented Java language. From the layers of the classical architecture (cf. Figure 2.2), Java Applets offer the full Toolkit services and some Windowing System services (for instance, Applets can open new windows). According to Fielding, Java Applets have not succeeded at least partly because they do not fit the deployment model of web technology, and have relatively bad ease-of-learning [44]. The benefit of Applets is that the development language is relatively advanced, allowing good modularization and re-use, a theme especially important for complex user interfaces. Also, a single language can be used in the client as well as the server. The drawbacks are performance, device independence, and ease of learning, which are worse compared to the declarative approaches.

Java Based Web Toolkits

A recent approach to web UI tools are the Java Based Web Toolkits. The main idea is that the whole user interface is developed in Java, without actually using web technologies. The server-side component then transcodes the user interface into DHTML or Ajax, which can be run in a normal browser without a Java virtual machine.

Two examples of this approach are Google Web Toolkit (GWT)[1] and the open-source Echo2 framework[2]. The Google Web Toolkit even allows the client side code to be run in a debugger, thus making development easier. Therefore most of the benefits and drawbacks of Java Applets apply to this approach. Compared to Java Applets, this approach has better web integration, since the client-side code runs in a normal web browser.

3.2 Declarative UI Tools

Authoring UIs with toolkits and procedural languages is acceptable, even beneficial, in the case where the computing ecosystem is homogeneous and applications not distributed, but this will not be the case in the future, where multitude of different devices are being utilized to access the web applications. This has shifted the focus of user interaction research from toolkits and components towards declarative languages, where it is easier to raise the semantic level and allow adaptation to different devices and contexts. The basic difference between procedural and declarative languages is that when procedural languages tell how to do things, declarative languages try to capture the idea of what to do.

Schmitz has studied multimedia description languages, and provides three main reasons why declarative languages should be preferred instead of procedural languages [102]:

Authors are not programmers Bulk of current web authors are not programmers, and if a

[1] Google Web Toolkit. Software. Available at: *http://code.google.com/webtoolkit/*, Accessed Nov 2006.

[2] NextApp Echo2 Java Web Toolkit. Software. Available at: *http://www.nextapp.com/platform/echo2/echo/*. Accessed Nov 2006.

Table 3.1: Comparison Between W3C Recommendations.

	XHTML	**SVG**	**SMIL**	**CSS**	**XForms**
Multimedia Objects					
Video	<object>	-	<video>	–	–
Audio	<object>	<audio>	<audio>	–	–
Text	Yes	Yes	<text>	–	–
Images					
Bitmap	<object>	<image>	<img>	Background	–
Vector	<object>	Yes	<img>	–	–
Visual Style	–	Yes	-	Yes	–
Object Arrangement					
Spatial	Flow	Absolute		Flow and abs.	–
Temporal	-	-	Yes	–	–
Animation	-	Yes	Yes	–	–
User Interface State					
Language	-	-	-	–	Yes
Scripting	Yes	Yes	-	–	Yes
Submission	Forms	-	-	–	Yes
Interaction					
Presenting	Yes	Yes	Yes	–	Yes
Scrolling	Provided by User Agent				
Links	<a>	<a>	<a>	–	Yes
Higher	Forms	-	-	–	Yes

description languages would require programmers, the target audience would be much smaller.

Authoring tools interoperability It is nearly impossible to create interoperable visual authoring tools for non-declarative languages.

Adoption in marketplace Without a standard, declarative solution, the authoring and publishing environment becomes fragmented, which ultimately hurts the adoption and advancement of technology. While not a technical requirement, this is an important point-of-view.

Almost all current declarative UI languages are XML-based, even though there is a long tradition of pre-XML languages (cf. e.g., [47, 51]). XML was originally defined by W3C, which is still the authoritative body for web standards. Some XML-based W3C recommendations are compared by the Author and Dr. Pablo Cesar in Table 3.1. The comparison shows that, currently, W3C is defining languages intended for specific purposes: eXtensible Hypertext Markup Language (XHTML) for document structure, SVG for vector graphics, SMIL for timing, Cascading Style Sheets (CSS) for visual style, and XForms for user interaction.

User Interface Description Languages

Since the advent of XML, there have been an ongoing "race" to define the ultimate XML-based UIDL, which would ideally capture the essence of what UI could be [109]. This Chapter tries to summarize the main approaches that have been proposed in the literature.

Puerta and Eisenstein propose a UIDL called *eXtensible Interface Markup Language (XIML)* [95], based on MIMIC [94]. It's requirements include central repository of data, comprehensive lifecycle support, abstract and concrete elements, and relational support. XIML supports modeling the UI at very abstract level, where the Task, Domain, and the User of the UI is modeled. On the lower level, it supports modeling the Dialog and the Presentation. XIML separates the data from the presentation well. A possible problem is that it does not support third party extensions, or synchronization of the UI state (e.g., submission) [112].

Universal Remote Console (URC) by the V2 Technical Committee of INCITS [3] is a UIDL targeted at the scenario of an universal remote controller [118, 119]. Although it naturally fits this scenario perfectly, it is unclear whether it could be extended to other usages as well. For instance, the run-time data model is quite limited.

Abrams et al. have proposed a language called *User Interface Markup Language (UIML)* [2]. It is one of the earliest XML-based proposals, and the current version is 3.0 [1]. UIML's design goals include separation of UI code from non-UI code, wide targeted author-base, rapid prototyping, extensibility, security, usability. In contrast to its design goals, it has been said that it does not allow the creation of user interfaces for the different languages or for different devices from a single description [109]. Also, it has been criticized of not separating user interface elements from their presentation and lacking an explicit data model [112].

Renderer IndependentMarkup Language (RIML) is a proposal for incorporating existing XML-based declarative languages (XHTML, XForms, and SMIL) into a profile, which has additional mark-up for pagination and layout for small devices [116]. The method falls in the *Unique portable code* / *generic client* category (cf. Section 2.5). It is also a single-authoring method with author defined rules for adaptation [105].

USer Interface eXtensible Markup Language (UsiXML) is a model-based approach for authoring context-sensitive adaptive user interfaces [69]. It has similar goals as XIML, and addresses all the layers of the Camelon reference framework (cf. Section 2.5). In contrast to many other approaches, in UsiXML, the developer can start the design work at any level of abstraction, which is considered its biggest strength [105].

(X)HTML

The term *Hypertext* was created by Ted Nelson in 1965 [81]. Few research systems in the 1960's already had some hypertext features [36, 80, 64]. However, hypertext became widespread when the WWW was created by Berners-Lee in 1990 [80, 9]. The original proposal for WWW included HyperText Markup Language (HTML) as the format to encode

[3] The V2 Technical Committee of INCITS. Available at *http://www.incits.org/tc_home/v2.htm*. Accessed Nov 2006.

content and links. HTML is a abstract document description grammar defined using Standardized General Markup Language (SGML).

Nowadays, XHTML [90], which is based on XML, is starting to replace HTML [96] as the document description format, but the underlying idea remains the same: simple markup, which can be handwritten, describes the structure, layout, and the content of web pages. This has made the WWW's *ease of learning* good for both the users and the authors of WWW, while still providing decent *expressivity* (how powerful is the tool), in the form of multimedia capabilities, layout, and interactivity. The interactivity in WWW is provided by links and HTML forms. HTML forms is a limited set of form controls or widgets, combined with a submission protocol, allowing the browser to send the filled contents of the form to the server, using a simple name-value pair mapping.

As depicted in Figure 3.1, most of the interaction in the original HTML model is done in the server. The client is quite thin; the only interactions, which are executed in the client are: scrolling, activating links, submission, and filling in simple form controls. Even a simple date validation is done at the server. The server works by generating a new HTML page for each user interaction, and there is no state in the client between the pages [44]. The problem in this approach is the low interactivity and high latency, since all UI logic is executed at the server.

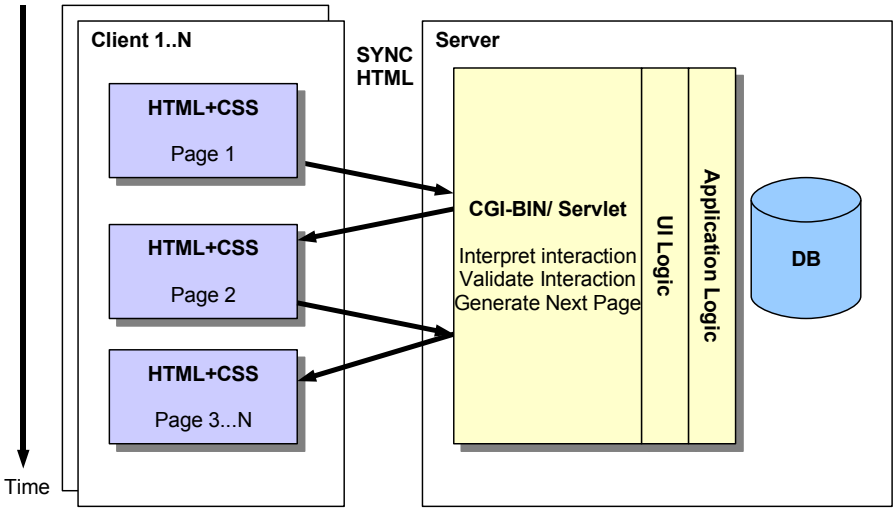

Figure 3.1: HTML interaction model

XForms

The XForms language, which is a W3C Recommendation [31], solves some of the problems found in the HTML forms by separating the purpose from the presentation and using declarative markup to describe the most common operations in form-based applications. It is actually a device and modality-independent UIDL, which is based on earlier form technologies, such as XFDL [12, 18] and XFA [74].

XForms separates the form into three main layers: Model, Instance, and User Interface

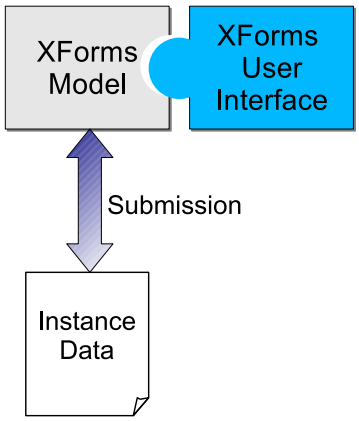

Figure 3.2: The XForms layers.

(cf. Figure 3.2).

Instance An arbitrary XML document that is modified by client side user interaction, then submitted to a server.

Model Uses XML to define the constraints on items of the instance, which includes data types and ranges as well as computational relationships.

User Interface Defines how the form is presented and expresses bindings to instance items. User input is governed by rules in the model for the instance item being modified (through a bound input control). This layer also includes more advanced functionality, such as repeating constructs, and dynamic UI bindings, which can be used to realize more complex UI patterns.

The Model layer includes the *XML Schema* and the *Model Item Properties*. The structure and the data types of the instance data can be defined using the schema, but it's use is optional to the UI author (Dubinko shows that some datatypes are more useful than others in XForms [29]). The Model Item Properties (MIP) are dynamic constraints written in XPath. They can reference other values in the instance data and are dynamically evaluated by the XForms Processor. With MIPs, it is possible to define dynamic calculations and cross-value checks, which are not possible with XML Schema. The evaluation model of MIPs, which involves automatically determining the calculation order [P3], is similar to that of spreadsheets [17].

Compared to HTML, a lot of the user interface processing is transferred to the client in XForms. This frees bandwith and improves the user experience. Additionally, the application data is separated from the user interface, which helps to build modular software, which is easier to maintain. The XForms interaction model cleanly separates the user interface logic from the application logic, as depicted in Figure 3.3. The UI logic is described declaratively, and to obtain optimal usability and network-traffic, the UI logic can be completely executed at client by an XForms processor. The XForms language allows to specify asynchronous communications with the server, and while the asynchronous request is being processed, the rest of the user interface remains responsive.

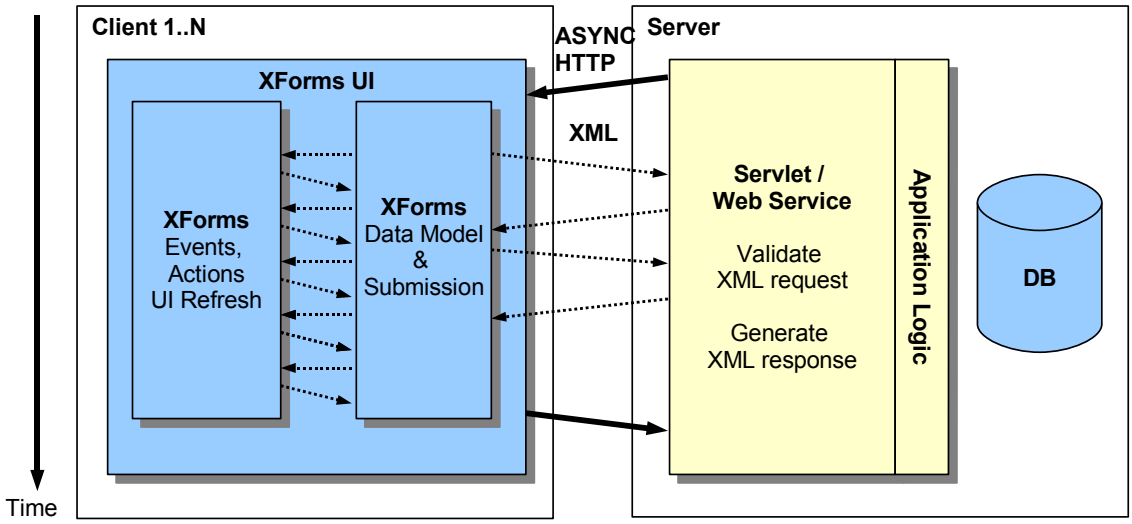

Figure 3.3: XForms interaction model

UI Bindings in XForms allow dynamic dependencies in the binding expressions. For example, consider the UI binding in the following simplified example:

```
1: <select1 ref='/ui/curr' />                    (master)
2: <input ref='person[@id='/ui/curr']/birth' />  (detail)
```

The input control in line 2 is bound to a person's birthday, but the predicate adds another feature: the select1 control in line 1 determines, which person's birthday will be edited at given time. This allows easy creation of so-called master-detail user interfaces, where a master selection shows the item titles, and selecting an item reveals its details, which can then be edited. The process of changing the currently bound UI node is called *rewiring*, and it happens automatically when processing the xforms-refresh event.

Because of the quite heavy dependencies on XML processing tools, such as XML Schema and XPath, XForms may be difficult to implement in a restricted device, such as a smartphone. To overcome this problem, W3C has specified a smaller profile of the language, called XForms Basic [30], which does not include full Schema capabilities.

The XForms 1.0. Second Edition [41] is a specification that fixes the errors found in the first edition and adds support for more flexible partial replacement of instance data. The XForms 1.1 Working Draft [40] is the next major version of the XForms language. It adds, among others, new XPath extension functions, support for manipulating structures, conditional actions, iterations, non-textual media output, and better SOAP integration.

There are different deployment options for XForms. These are described in the following list:

1. **Native browser support.** If the user agent supports XForms natively, the XHTML + XForms UI descriptions, along with possible XML schemas, are served as static documents to the user agent. The instance data can be dynamically generated at the server

and serialized as XML. Similarly, submissions from the user agent are received as XML by the application. An example of such approach is X-Smiles [P1].
Pros: Very low UI latency. Simple server deployment. The browser is the only trusted component. Low bandwith consumption
Cons: Currently only supported in some experimental user agents.

2. **Browser plugin.** If the browser has a rich plugin interface, it is possible to build the XForms client as a browser plugin. An example of such plugin is formsPlayer[4], which utilizes the behaviors API of Internet Explorer. The Mozilla XForms implementation falls into this category as well. After the installation of the plugin is done once, the application can proceed as in option 1.
Pros: Very low UI latency. Simple server deployment. Low bandwith consumption.
Cons: Only supported in some user agents. Have to trust plugin vendor.

3. **AJAX implementation.** Since all mainstream browsers support AJAX, it is possible to create a full-blown XForms implementation in AJAX. This way it is possible to run client-side XForms interactions without installing any additional software in the client. The application can proceed as in option 1 by adding a reference to the AJAX library to each UI description. This approach is demonstrated by the open-source FormFaces[5] processor.
Pros: Simple server deployment. The browser is the only trusted component.
Cons: From low to higher UI latency. Have to transfer the XForms library along with the UI description (higher bandwith consumption).

4. **Server-side transformation.** XForms+XHTML UI descriptions are transformed into plain XHTML or DHTML by a server-side process. The submissions can be done, e.g., by AJAX or using HTML forms submissions and transforming them into XML in the server. This deployment option allows XForms to be used without installation of special plugins in user agents even without ECMAScript support. This approach is used in the open-source Chiba[6] and AJAXForms[7] processors.
Pros: The browser is the only trusted component.
Cons: From low to higher UI latency. Complicated server deployment. Higher bandwidth consumption.

3.3 Hybrid UI Tools

Hybrid UI tools is a category of languages, which have properties both from the declarative and procedural tools. The hybrid tools usually combine a declarative language with a

[4] xPort FormsPlayer, Software, available at *http://www.formsplayer.com/*, Accessed Nov 2006.
[5] FormFaces, Progeny Systems, Software, available at *http://www.formfaces.com/*, Accessed Nov 2006.
[6] Chiba, Software, available at *http://chiba.sourceforge.net/*, Accessed Nov 2006.
[7] AJAXForms, Open Source Software, available at *http://ajaxforms.sourceforge.net/*, Accessed Nov 2006.

procedural scripting language, such as ECMAScript[8].

DHTML

Since the HTML interaction model has problems related to usability and scalability, a more advanced model has been created. This model is called *Dynamic HTML*. It contains three main components: declarative description with (X)HTML + CSS, interface for dynamically modifying the document (DOM Level 2 Core and Events) [37, 39], and procedural scripting (ECMAScript) [33]. This model (cf. Figure 3.4) allows to execute user interaction logic at the client. This is extremely important for lowering the latency, since there can be large amount of clients using the application simultaneously. If every interaction, such as validation of an input, or filtering a view has to be implemented in the server side, usability and scalability will suffer. Currently, DHTML is implemented in all major browsers.

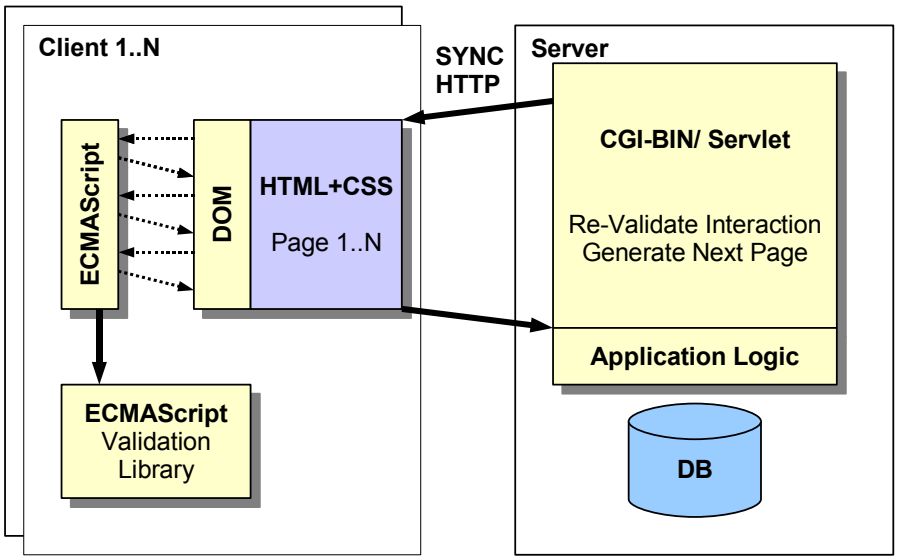

Figure 3.4: DHTML interaction model

WhatWG Web Forms 2.0

There is another proposal, called Web Forms 2.0[9], for extending HTML forms by the industry working group WhatWG. It extends DHTML forms by adding basic data types, client-side validation, and a repetition model. Web forms does not have a data model, nor it separates the data from the presentation. The main benefits over DHTML are better ease-of-authoring, and increased accessibility and device and modality-independence (because of added semantic information in the form of datatypes and higher-level constructs).

[8] In this Thesis the term ***ECMAScript*** is used as a synonym for JavaScript, which is the commercial name for the scripting language.

[9] Web Forms 2.0, WhatWG Working Draft, 12 October 2006. Available online at: *http://www.whatwg.org/specs/web-forms/current-work/*. Accessed Nov 2006.

The main approach of Web Forms 2.0 has been to take the XForms requirements and fullfill them in a way, which integrates easily to DHTML. The main differences to XForms is the lack of structured data model and the heavy reliance to procedural scripting, while XPath is used in XForms.

At the time of writing, Web Forms is supported by Opera 9.0 and an ECMAScript implementation, which works in Internet Explorer[10].

AJAX

Recently, an even more interactive model has been taken into use. This model is called Asynchronous JavaScript and XML (AJAX) [49], and it extends the DHTML model. In addition to (X)HTML, CSS, DOM, and ECMAScript, it uses an XMLHTTPRequest ECMAScript object, which enables asynchronous, behind-the-scenes, XML submissions over HTTP [89]. Asynchronous submissions fix the interaction problem of DHTML, by allowing the user to continue interaction while a submission to the server is being processed.

Compared to DHTML, it is possible to create lower latency UIs with with AJAX, since the UI remains responsive while partial submissions are being made. Typically an AJAX application uses libraries, written in ECMAScript, to handle typical tasks, such as serialization, deserialization, submission, validation, and interaction. Integration between the declarative XHTML and the libraries is handled through DOM Events and ECMAScript handlers.

Compared to an declarative UIDL, such as XForms, the main problems in AJAX are similar to those of DHTML: lack of semantics in the model, problems in accessibility, maintenance, and ease of authoring. All of the interaction and communications logic is written in procedural scripting language, which is harder to statically analyze or adapt automatically, compared to the declarative approaches. Other problems include limitations of the basic ECMAScript language, browser incompatibilities; and a lack of tool support for designing, developing, and debugging this new breed of web application [107].

Currently, AJAX is implemented, at least to some extent, in most of the major browsers, and examples AJAX applications are Google Suggest[11] , and Google Maps[12]. For instance, Google Suggest is an interface to a web search engine, which suggests search terms interactively with each keypress. This is done by sending asynchronous HTTP requests to the server for possible suggestions.

For all these models (i.e., HTML, DHTML, and AJAX), there exist many tools, which can be used at the server side, including JSP, ASP, Servlets, and Struts. One problem is that the applications become a mixture of markup languages, client-side scripting code and server-side function calls, making them harder to maintain [23].

[10]Web Forms 2.0, Software. Available online at: *https://sourceforge.net/projects/wf2/*. Accessed Nov 2006.

[11]Google Suggest. Application. Available online at *http://www.google.com/webhp?complete=1*. Accessed Nov 2006.

[12]Google Maps. Application. Available online at *http://maps.google.com/*. Accessed Nov 2006.

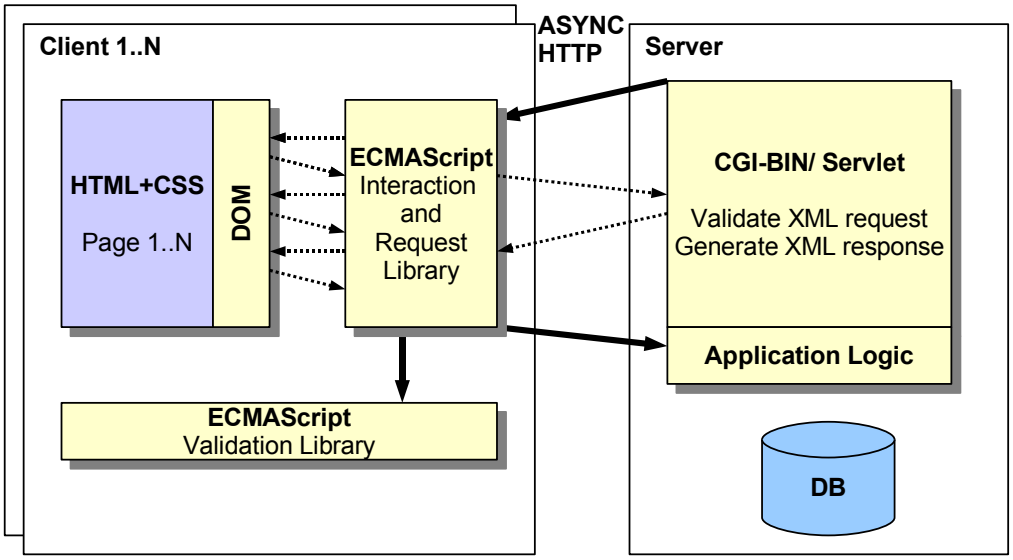

Figure 3.5: AJAX interaction model.

XUL

Hyatt et al. have developed a UI description language called *XML User Interface Language (XUL)* [34]. XUL is targeted for desktop use, and it includes notions of windows, panels, dialogs, along with all the typical desktop widgets. XUL itself does not have an data model, validation, or calculation facilities, but it is typically used in conjugation with XBL [58] and ECMAScript [33] to provide these.

XUL was designed for defining the user interface of Mozilla (or Netscape) browser. Compared to more generic UIDLs, such as XForms, device and modality independence is therefore worse. Also, the user interface logic has to be programmed using a scripting language. Maybe the biggest difference is the communication between the user interface and the back-end system. For the communication, the user interface state has to be serialized, similar to AJAX, for transmission. Conversely, after getting serialized reply from the server, it has to be deserialized into application state. In XForms that serialization is automatic, since the datamodel is a live XML document object model, which is automatically serialized and de-serialized. On the other hand, in XUL there is no explicit datamodel, and communication between a backend process and the user interface have to be reimplemented using ECMAScript for each user interface. This means, that ease-of-authoring in XUL is worse compared to XForms. It is noteworthy that XUL has a templates mechanism, which allows to use RDF as the datamodel to some extent, although RDF is more complicated than XML (graph vs. tree). Using XBL [58] with XUL allows the use of restricted XML datamodels [P8].

Other Hybrid Tools

Finally, there is currently a lot of effort put into XML-based desktop GUI technologies, including Glade[13], Netscape XUL [34] and Microsoft XAML [100]. On the other hand, InfoPath [55], addresses the office application space. These desktop UI languages are typically combined with a procedural language for UI logic implementation, thus falling to the hybrid UI tool category.

3.4 Comparison of Approaches

As described in the earlier Sections, the reviewed UI tools fall into three categories (i.e., procedural, hybrid, declarative). Also, some tools are derived from others, such as DHTML, which is derived from HTML by adding scripting. This is depicted in Figure 3.6. The main difference between the categories is the semantic level, which is typically highest in the declarative tools, and lowest in the procedural tools.

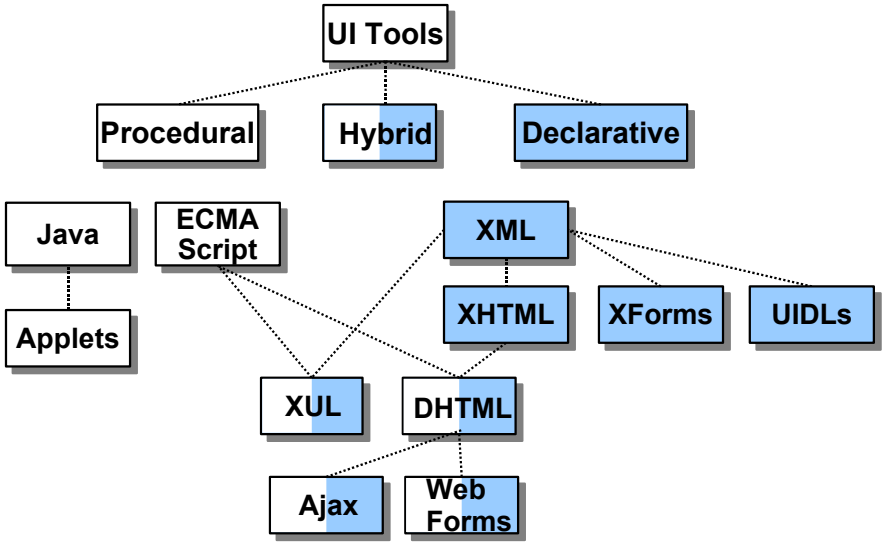

Figure 3.6: The relationship between the compared UI tools.

Souchon and Vanderdonckt have conducted a review of XML-based UIDLs [109]. The compared languages, were UIML [1], XIML [95], AUIML [5], Seescoa XML [70], AAIML / URC [118, 119], Teresa XML [88], WSXL [3], XUL [34], XISL [60], and Tadeus-XML [77]. The conclusion in that paper is that the (multitarget) expressivity differs a lot, XIML having the highest and UIML the lowest. Also, XIML is extendable, the downside being the license that is needed in order to utilize it, while most of the other languages can be used freely. Also, XUL and other lower-level languages have better implementation support. XUL is not considered a “genuine and complete” UIDL, and it is said that it only addresses “some requirements for supporting multiple platforms” [109].

[13]Glade. UIDL and software library. Available at: *http://glade.gnome.org/*. Accessed Nov 2006.

Trewin et al. have also reviewed XML-based UIDLs from the point of view of device independence (or universal access). Their review includes UIML [1], XIML [95], XForms [43], and URC [118, 119]. They conclude that XForms and URC, fits their requirements best from the reviewed languages [112], while the others fail in different ways. They compare languages in terms of applicability to any target, delivery context, personalization, extensibility, and simplicity. Most of their work is applicable to web context, as well.

Simon et al. conclude that XForms is similar than their earlier research results [105] on multimodal UIDL. They claim that this validates XForms for multimodal usage. Also, the use of XForms as the UIDL can simplify web application development, since instead of generating HTML markup and embedded client-side scripting in a server side program, it is possible to use purely declarative XHTML+XForms description of the user interface, thus separating the client and server-side concerns [23].

The reviewed tools are ***compared against the proposed criteria*** in Table 3.2. In that table the following items contain ECMAScript in central role, thus belonging to the *hybrid* group: DHTML, Web Forms, Ajax, XUL, SVG, Ajax+SVG+WebForms 2.0. Java Applets and Java web toolkits belong to the *procedural* and the rest to the *declarative* group. Note that the table is based on the Author's current knowledge on the state of the art, and is thus somewhat subjective.

Few trends are noticeable from the table; expressivity and low latency are highest in the procedural category, and lowest in the declarative category. On the other hand, device- and modality- independence are higher in the declarative category, namely in XForms. This is in line with the findings of Simon et al [105]. It is evident from the table that not a single language fullfills all criteria well. Other UIDLs, such as UsiXML, XIML and UIDL, are not compared in this table, since they are reviewed in literature [109, 105]. The strength of some of these UIDLs, such as UsiXML, is that they have layers that reach higher semantic levels, such as Tasks & Concepts (cf. Section 2.5). On the other hand, their use may require a different skill-set, compared to a typical programmer or web UI author, or sophisticated (often missing) tools, and they have not yet proven themselves outside the research community.

In this Thesis, XForms was chosen as the basis for user interaction description. The main technical reasons were the requirements for XForms to support device independence, ease-of-authoring, and higher-level UI definitions, while still being expressive enough for rich internet applications [112, 105, 23]. Other reasons were related to the fact that W3C was defining XForms, thus assuring the quality, interoperability, and the widest audience, and that the Author was participating the specification work as an invited expert, thus being able to affect the development of some key features.

Based on the Table 3.2, it can be claimed that a combination of Ajax, SVG, and WhatWG Web Forms 2.0 could be almost as good solution based on the criteria. There would still be some drawbacks since Web Forms 2.0 does not provide transmission of end-to-end structured data and declarative interdependencies of data and the view.

Table 3.2: Comparison of the tools reviewed in this Thesis. Empty cell means less than average, '+' means average, and '++' means better than average. *) These include procedural component (ECMAScript or Java) in central role.

	General Criteria							Web Specific						
	Expressivity			Authoring										
Tool	Visual Presentation	Interaction	Multimedia / Synchron.	Ease of learning	Ease of authoring	Extensibility	Authoring tool interop.	Device Independence	Modality Independence	Low latency	Web Integration	Accessiblity	Security	Ease of implementation
Applets*)	++	++	+			++				++			++	
WebTK *)	++	++	+		+	++				++	+		+	+
DHTML*)	+	+	+			++		+		+	++		+	+
Web Forms*)	+	++	+	+	+	++	+	+	+	+	++	++	+	+
Ajax*)	+	++	+			++		+		++	++	+	+	+
XUL*)	+	++				+				++		+	+	+
SVG*)	++	+	+			+	+			++	+		+	+
AjaxSVG WForms*)	++	++	++			++	+	+	+	++	++	++	+	+
XHTML	+			++	++		++	+	+		++	+	+	++
SMIL	+		++	++	++		++	+	+	+	+	+	+	++
XForms	+	++		+	++		++	++	+	++	++	++	+	+

4 PROPOSED MODEL FOR NEXT-GENERATION WEB INTERACTION

This Section describes the high-level contributions of this Thesis; the proposed model for next-generation web interaction. It can be applied regardless of specific device, operating system, or user agent implementation. Implementation-specific results are described later in Chapter 5. For each topic, requirements are presented followed by the proposed solutions.

In order to fullfill the current demands (c.f. Chapter 1), and to make sure that future demands can be addressed as well, the model is layered and extensible. It is divided into modality-independent and modality-specific parts (cf. Figure 4.1). The modality-independent part includes a data model, and abstract UI layers. The modality-specific part contains stylesheets, timing, and custom controls. The arrows in the Figure denote the direction of dependency (e.g., stylesheets are dependent on the abstract UI layer). The details of these layers are provided in the following Sections in this Chapter.

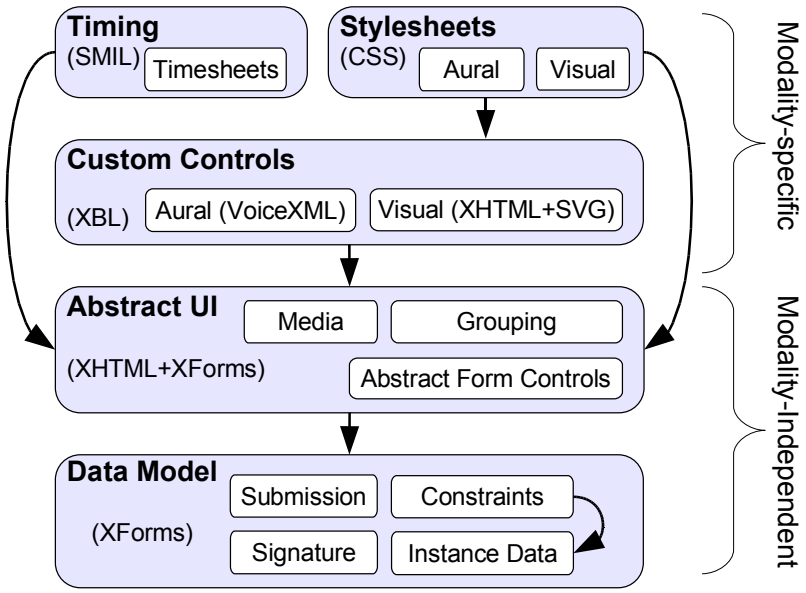

Figure 4.1: The Proposed Layered Model.

4.1 User Interaction with XForms

Next-generation web applications will require much higher level of user interaction compared to the current ones. Therefore interaction has been the focal point of this work. The most comprehensive requirements for abstract UI representation have been derived by Trewin et al. [112], based on the work of Nichols et al. [82] and Zimmerman et al.

[117, 118]. These are used as the basis for the set of requirements for user interaction. The publication [P8] extended these requirements, taking into account the application scenarios (c.f., Section 2.2), and these are extended even further here.

Requirements: User Interaction

1. *Layer separation*
   (a) *Separation of interface elements from presentation* [112]. Allows easy re-use and replacement of different parts.
2. *Interface elements* [112]
   (a) *Dependencies between elements.* It must be possible to declare that the state of a user interface element is dependent on the state of another element.
   (b) *Applicable to any target.* The UI definition must be interpretable on any target device.
   (c) *Data types*. The data items should be typed, so that it possible to generate concrete UI for any context of use.
3. *Presentation related information* [112]
   (a) *Logical groupings*. Instead of graphical layout, logical groupings should be preferred, allowing for adaptation based on the target and context.
   (b) *Labels and help text*. Labels and help texts should be easily replaceable for internationalization and adaptation.
   (c) *Presentation replacement*. The entire presentation, or parts of it must be possible to replace in order to support personalization and adaptation.
4. *Run time and remote control*
   (a) *State representation* [112]. The user interface state should be explicitly available.
   (b) *Local computation* [112]. Client-side computations should be possible to be represented by the UI definition.
   (c) *Synchronization.* It must be possible to synchronize the user interface state between the client and an external process. In web context, this typically means client-side submission, but other type of communication methods should not be ruled out [P8].
5. *Typical interaction patterns*. Typical interaction patterns or idioms should be supported by the UIDL (extended from [P8]). Note that most of these patterns require dynamic UI, either by modifying the dialog flow or by changing the underlying binding of the model and view parts. This list of conceptual UI patterns is by no means exhaustive, but should be representative considering the use case scenarios in Section 2.2.

(a) *Typical interactors*. Must support current set of user interface interactors, such as selections (one or many) from a list, text input, triggers, etc.

(b) *Master-detail*. A type of UI where a list of items, and details of the selected item are shown [76].

(c) *Paging and dialogs*. Must be possible to split the form into multiple pages, while retaining dependencies. Different kinds of paging representations should be allowed, such as wizards and tabbed panes.

(d) *Repeating constructs*. Navigating and editing repeating data sets. These sets can be represented e.g., as dynamic tables.

(e) *Nested constructs*. Navigating and editing nested data sets.

(f) *Copy-paste*. Copying a data item from one part of the UI to another part.

(g) *Undo-redo*. Undoing and redoing the last actions.

(h) *Drag-and-drop*. Interactively moving a piece of information from one place of the UI to another.

(i) *Context menus*. Showing and letting the user select actions that are applicable to the selected object.

(j) *Filtering.* Provide the user a mechanism to filter a large dataset. Usually the user is able to make a choice among the reduced dataset [76].

XForms [31] is proposed as the language for user interaction, based on the reasons explained in Section 3.4. The Table 4.1 summarizes how XForms addresses the requirements 1-4. Further explanations are provided in the following list:

- *Separation of interface elements:* Separation of UI data and presentation is supported well, also the re-use of data models in different presentations is supported [112]. The XForms model is a pure data model without any presentation semantics.

- *Interface elements:* The data model is sophisticated, includes data constraints, and is exposed explicitly. It is modality-independent, as well. The use of declarative expressions improves to quality of generated user interfaces [112]. Note that Trewin et al., seem to have misinterpreted XForms to allow XPath functions to call scripts, while in reality this is not supported. In order to describe dependencies between data items in the model, a calculation engine is specified [P3].

- *Presentation information:* XForms UI provides abstract presentation widgets, along with semantic elements to include labels and help messages [112]. Note that Trewin et al., claim that replacing labels defined in instance data would require greater flexibility, although in reality, all that is needed is to replace a single externally referenced XML instance document.

- *Run time and remote control in XForms:* One way (client-initiated) synchronization is supported through submission, while target-initiated (i.e., server side) synchronization is not supported [112].

Table 4.1: Solutions for interaction requirements.

Requirement	Solution
1a) Separation of interface	**XForms 1.0.** The **model** separates the form data from the presentation of the form.
2a) Dependencies	**XForms 1.0.** Interface elements are bound to the model with **UI bindings**, and can have dependencies through the **model**.
2b) Any target	The **abstraction level** in XForms is high: it is possible to interact with the UI in different target devices, and even with voice (cf. Publication [P8])
2c) Data types.	XForms fully supports **XML Schema datatypes**, and form controls are adapted to the underlying datatype.
3a) Logical groupings.	**XForms 1.0.** The **group, case**, and **repeat** elements group together interactors.
3b) Labels and help text.	The elements **label, hint, alert** and **help** can provide multilingual help texts for different situations.
3c) Presentation replacement	**CSS, Relevant MIP**, and **Groups** provide different ways to replace parts of the representation.
4a) State representation.	**XForms 1.0.** The **instance data** captures the current state of the form. It can easily be submitted and later restarted.
4b) Local computation.	XPath statements in **MIPs in bind** element is used to provide local computation between the items in the instance data.
4c) Synchronization	The **submission** element provides the rules to submit the Instance data to a back-end process.

The Table 4.2 lists the solutions to the *typical interaction patterns* requirements. XForms provides a list of *typical interactors*, including triggers, selection lists, and input. These interactors are datatype-aware, which means that an implementation should support at least the XML Schema basic datatypes, including e.g., date and time. There is two ways of extending this set of available interactors: First, implementation specific interactors for specific new datatypes can be supported. Second, this Thesis proposes to use XBL for author-defined interactors (cf. Section 4.5).

XForms has been designed so that it allows dynamic interaction patterns locally, in addition to the static forms, which are typically supported by UIDLs. *Paging and dialogs* is provided by switch-case construct [RP2] *Repeating constructs* are provided by the repeat element [RP2], which allows building any repetitive view supported by the host language (e.g., CSS tables). *Master-detail*, and similar dynamic patterns are provided by dynamic UI bindings, which mean that the binding between the model and the UI part can change runtime. Also, the MIPs provide a way to make items disabled, read-only, or disappear based

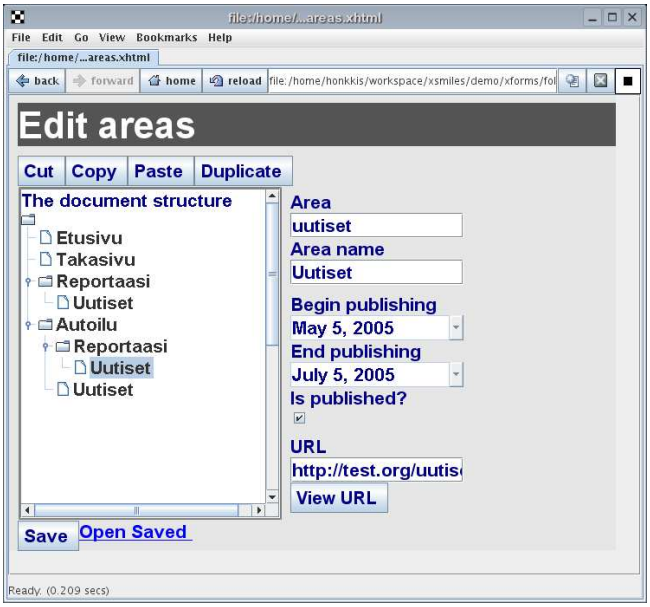

Figure 4.2: An example of the tree module usage [P8] to provide master-detail form for editing structured data. The dynamic interaction is handled completely locally.

on user selections. For instance, an OK trigger can be activated or deactivated based on a previous value selected by the user through the use of the relevant MIP.

It is noteworthy that XForms allows editing and displaying repeating collections, but the nesting level has to be specified by the UI author. Indefinitely deep *nested constructs* are not supported. This is a serious drawback in many user interfaces. Typically a nested structure is presented as a foldable tree widget in a graphical UI. Therefore an extension to XForms is proposed by the Author in the Publication [P8], which extends XForms 1.0 specification with a tree module containing a new module *xforms:tree* and a corresponding XPath extension function *nodeindex*. For example, the extension allows easy authoring of master-detail UIs where the master is a recursive structure of indefinite depth (similar to a file system explorer).

As a summary, XForms fits most requirements well, except for the two-way synchronization, and some typical interaction patterns. For drag-and-drop, an event-based solution, similar to what What WG Web Applications Working Draft[1] proposes, should be added to XForms. Context menus have several proposals in the W3C www-forms mailing list[2], and one of them should be picked up. In the web context, two-way synchronization is not supported by the underlying transport (HTTP). In some use cases, it is emulated using asynchronous polling. For instance, Gmail[3] uses AJAX HTTP polling to check for new incoming chat and email messages. Polling requires asynchronous HTTP requests, which are also sup-

[1]WhatWG Web Applications 1.0 Working Draft 26 November 2006. Available at: *http://whatwg.org/specs/web-apps/current-work/*. Accessed Nov 2006.

[2]www-forms, W3C mailing list. Archive available at: *http://lists.w3.org/Archives/Public/www-forms/*. Accessed Nov 2006.

[3]Gmail, Web application. Available at *http://www.gmail.com/*. Accessed Nov 2006.

Table 4.2: Solutions for typical interaction patterns requirements.

Requirement	Solution
5a) Typical interactors	**XForms 1.0 - input, output, trigger, select1, select, secret, textarea, upload, range, submit.** XForms supports typical interactors through these elements. The interactor set can be extended, e.g., through datatypes.
5b) Master-detail	**XForms 1.0 - Dynamic UI Bindings.** The master-detail pattern is addressed by XForms natively by allowing the use of dynamic dependencies in UI binding XPath statements.
5c) Paging and dialogs	**XForms 1.0.** The **switch, case, and toggle** elements enable purely UI-driven switching of parts of the UI. This **relevant MIP** allows for model-based switching of parts of UI. Using **predicates in group** binding allows for model-based switching.
5d) Repeating constructs	**XForms 1.1.** The **repeat** element, associated pseudo-elements **::repeat-index** and **::repeat-item**, and the **index()** function enable creation of dynamic repeating constructs.
5e) Nested constructs	**Proposed Extension: tree, nodeindex()**: The tree extension proposed in Publication [P8] supports navigating and editing indefinitely nested data set.
5f) Copy-paste	**XForms 1.1 - insert, trigger:** Copy-paste at widget level is usually supported by implementations, and it is possible to create high-level copy-paste using triggers and insert statements in repeating data sets (cf. Publication [P8]).
5g) Undo-redo	Only **form control level** supported by some XForms implementations.
5h) Drag-and-drop	Not supported by XForms.
5i) Context menus	Not supported by XForms.
5j) Filtering	**XForms 1.0.** XPath predicates are used to provide dynamic filtering.

ported by XForms.

4.2 Visual Presentation with Dynamic CSS, SVG, and Timesheets

An important property for both multimedia presentations, as well as interactive user interfaces, is the visual presentation cababilities, which can be divided into arrangement of objects, and visual style. Furthermore, the arrangement of objects can be divided into spatial and temporal dimensions. The spatial layout makes it possible to group items, which are related to each other, thus giving semantic information to the user. Also, it is often necessary to have some items presented nearby, e.g., in the same screen. Spatial layout also contributes to the visual pleasantness of the user interface. Boll defines three possible layout models used in multimedia presentations: *Absolute Positioning*, *Directional Relations*, *Topological*

Relations [14]. An equally important layout model is the one used extensively in the web: *Text Flow* [101, 91]. These are depicted in Figure 4.3. The support for these models in web languages varies. For instance, SVG uses absolute positioning, while CSS has mixture of absolute, topological, and text flow positioning.

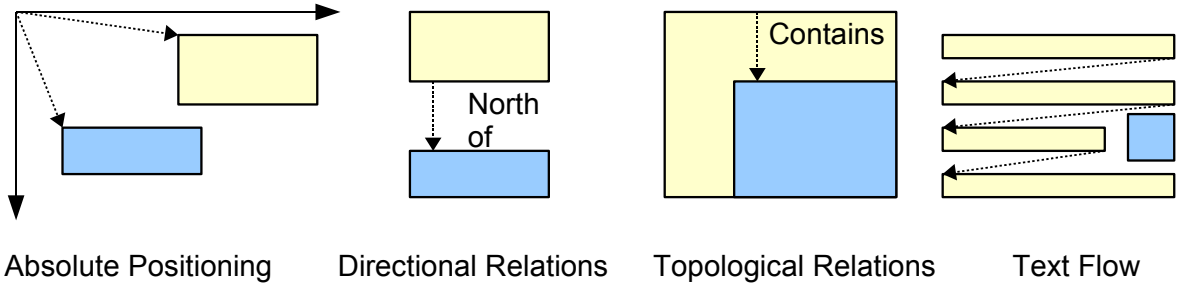

Figure 4.3: Different layout models [14, 101, 91].

Typically, the flow layout model is used in web applications, while a mixture of topological and directional relations (e.g., Java BorderLayout) is used in desktop applications. This has probably it's historical reasons: web started as a document repository, where flow layout was natural, while desktop applications typically used the other layout strategies in containers, with control over scrolling, within resizable, overlapping windows. On the other hand, current desktop environments, such as Windows, KDE, and Gnome have browser component, which is often used in applications, which require hypertext features. Similarly, CSS allows to build desktop-style layouts. Mixing the layout models is thus supported from the technology point-of-view (e.g., in CSS or Java), but it requires a bit more skilled developer.

Presentation requirements for multimedia applications have been extensively studied The following list of requirements is derived from literature by the Author together with Dr. Pablo Cesar.

Requirements: Visual presentation

1. *Multimedia Objects Support* [66, 15, 54]
   (a) Continuous Media: such as video and audio.
   (b) Discrete Media: such as text, graphics (bitmap and vector), and images.
2. *Visual Style* [113, 54]: authoring control over the visual appearance of the objects (e.g., colors, fonts).
3. *Arrangement of the Objects*
   (a) Temporal Dimension [21, 113, 15]: defines when the multimedia objects in a presentation should be presented / removed.
   (b) Spatial Layout [21, 113, 15, 54]: defines where the multimedia objects in a presentation should be placed.

(c) Animation [91]: combination of spatial layout and temporal dimension, in which the layout of the application is modified based on time.

The proposed solution for enabling rich presentation combined with declarative interaction language, is to use *Compound XML Documents* [75], which means that several XML languages are combined together in a well-specified manner. Two main ways to do this are a) *by reference* - use URL references from the host document or b) *by inclusion* - allow combining languages in a single document. The publication [P2] describes an early effort to provide a browser framework for combining XML documents by inclusion. The idea is that there are two kinds of XML languages, *hosts* and *parasites*. There is always exactly one host language in a document (e.g., XHTML or SVG), and there can be numerous parasite languages within one document. Also, a language can be both host and parasite. The paper [P2] describes some general interfaces, which are needed in order to communicate between the numerous languages mixed within a document. At the time of writing, a W3C has produced a draft of compound documents by reference [75] and requirements for compound documents by inclusion[4].

To describe the structure, layout, and styling of the interactive services, the use of XHTML together with CSS is proposed in the publication [P4]. XHTML is a well-established standard for describing document structures, while CSS provides a very powerful layout and styling model for XML (including XHTML) documents. Compared to other layout models, which support only batch creation of the final form document (e.g., XSLT+XSL FO), CSS supports higher user interaction: changes in the document (e.g., based on interaction) are easily reflected by using incremental rendering [68].

The publication [P4] also describes in detail the integration of XForms and XHTML+CSS, thus providing proof-of-concept for the XHTML 2.0 Working Draft, which is still a work in progress [42]. The multimedia objects (audio, video, and images) are included in the documents using the Object module of XHTML. HyperText linking is provided by the Link module.

Publication [RP4] proposes a way to include interaction in SMIL. It describes how to use CSS within SMIL to allow groupings of repeating form controls. For instance, one of the use cases is an exam results editor, which is created using XForms repeat functionality described in the Publication [RP2].

As noted in the paper, there are problems related to mixing temporal and positional properties in the integration proposed in publication [RP4]. The current proposal of the Author is to take the separation of timing and presentation even further. This idea of Timesheets is presented in W3C Member Submission by the Author et al. in the Publication [RP5], based on original work of ten Kate et al. [111]. Timesheets re-use the SMIL 2.0's Timing and Synchronization module [57], but create a indirection layer using CSS selectors, thus avoiding the problems related to mixing temporal and positional dependencies. This is depicted in

[4]Compound Documents by Inclusion is part of the document: Compound Document Use Cases and Requirements Version 2.0, available at: *http://www.w3.org/TR/CDFReqs/*, accessed Nov 2006.

Table 4.3: Solutions for presentation requirements.

Requirement	Solution
1a) Continuous media	**XHTML object** element provides support for video and audio elements.
1b) Discrete Media	**Most XHTML elements** deal with discrete media, such as text and images. **SVG** is included for vector graphics and combined with XHTML as defined in [75].
2) Visual Style	**CSS visual properties**.
3a) Temporal Dimension	**SMIL Timing And Synchronization module** [57] integrated with XHTML and SVG, used with the **timesheet element**, as suggested by [RP5].
3b) Spatial Layout	Cascading Stylesheets **(CSS) layout model** and properties.
3c) Animation	**SMIL Basic Animations module** [57], used with the timesheet element, as suggested by [RP5].

Figure 4.4. The submission was commented by the W3C staff[5], who concluded that many groups within the W3C should take the submission into account.

As a conclusion, the proposed solution (cf. Table 4.3) for visual presentation combines XHTML and SVG using Compound Documents Framework, uses CSS for host language layout model and visual style. Temporal dimension and animations are provide by SMIL Modules (i.e., timesheets).

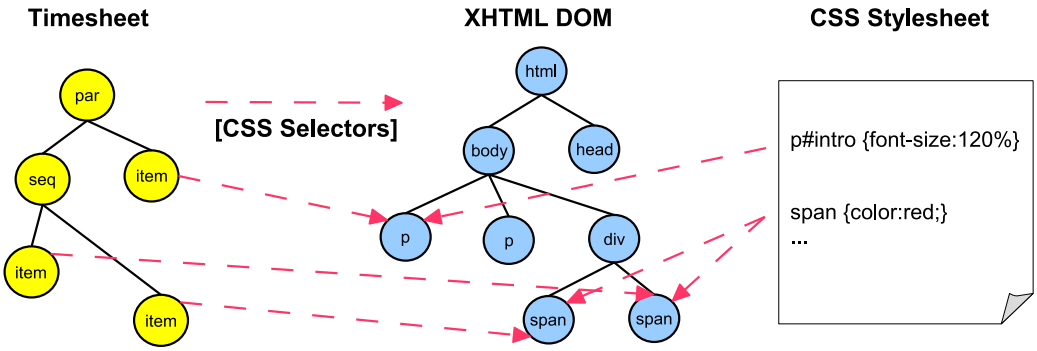

Figure 4.4: Timesheets reference presentation DOM using CSS Selectors.

4.3 Multimodal Interaction with XForms

In *multimodal interaction* two or more combined input modes are processed in a coordinated manner with multimedia system output. In many usage scenarios, it would be beneficial to allow the user to interact with multiple modalities simultaneously. Also, as applications have become more complex, a single modality does not permit the user to interact effectively

[5]Team Comment on the Timesheets: XML Timing Language Submission. Available at *http://www.w3.org/Submission/2005/03/Comment*. Accessed Nov 2006.

across all tasks and usage scenarios. An example of such a scenario is using different web applications with speech, while driving a car.

Technologies used in multimodal interfaces include conventional direct-manipulation devices like the keyboard, mouse, pen, and touch screen. Current, mature research integrates speech with pen, or speech with lip movement tracking. New research trends are towards more advanced input technologies such as speech recognition, 2D or 3D gesture recognition, lip movement, and gaze tracking.

Multimodal systems can potentially have many usability benefits over typical (unimodal) systems. Some of the benefits listed in the literature include:

Flexibility and Adaptability The user can flexibly choose which modalities to use based on the usage scenario. For instance, while driving a car, speech is the safest input and output method, since it leaves sight and hands free, and does not require full cognitive attention of the user. Multimodal interfaces also provide adaptation for mobile users with changing context. [86]

Accessibility An obvious benefit from multimodality is the increased accessibility. For instance, since the user interface is available in different modalities, a blind user can fully resort to speech, but for deaf user, different set of modalities is provided.

Intuitive and natural interaction People communicate multimodally with each other. For instance, gestures, facial expressions, and voice tonalities are important communication channels in addition to speech. This seems to hint that allowing multiple modalities when communicating with the computer would be of benefit [87].

Lower error rate Fusing inputs from multiple modalities together can compensate for the recognition errors through mutual disambiguation. In one study, this effect lowered the error rate compared to pure speech input by 19-35% [85]. Similar results have been found in other research [25, 84].

Efficiency Multimodal interfaces can increase efficiency since multiple channels of information are used simultaneously. This seems to be backed by experimental studies [86, 73, 38]. Although speech interaction seems to be promising, the potential technical problems may reduce task performance [104]. Thus, speech is often used by combining it with other modalities to offset the weaknesses of them [26].

Simon et al. have researched the area of authoring multimodal web applications [105]. They conclude that a cross-modal UI language, which provides *single-authoring* for multiple modalities would be the best approach. They compared their language, which was designed for modality independence, against XForms, and noted that the interactors were almost the same. They claim that this validates also XForms' modality-independence. There are also other proposals to realize multimodal interaction, such as XHTML+Voice profile [4] and SALT [115], which are evaluated in the Publication [P9]. Raggett and Froumentin propose extensions to Cascading Stylesheets (CSS) to allow simple multimodal interaction tasks [97].

The paper defines new CSS properties, which are used to add aural modality to current HTML applications. The aural UI is created separately through the CSS extension. These are reviewed in Publication [P9].

The publication [P9] also reviews and analyzes the requirements for Multimodal Interaction framework by W3C [71]. They are summarized below:

Requirements: Multimodal Interaction [71]

1. *General Requirements.* The integration of modalities should be seamless. Easiness to author, use, and implement a multimodal language are required. Finally, requirements include also accessibility, security, and delivery context related issues.

2. *Input Modality Requirements.* It is required that modality related information can be authored. The input should be able to be given sequentially, simultaneously, and combined from several modalities. Also, temporal positioning of input events should be available.

3. *Output Media Requirements.* Output must be able to be represented both sequentially and simultaneously in different modalities.

4. *Architecture, Integration, and Synchronization points.* It is desired to use compound documents with already existing languages. In addition, the specification should be modular. Data model, presentation layer, and application logic should be separated. Also, language should provide means to detect or prescribe the modalities that are available.

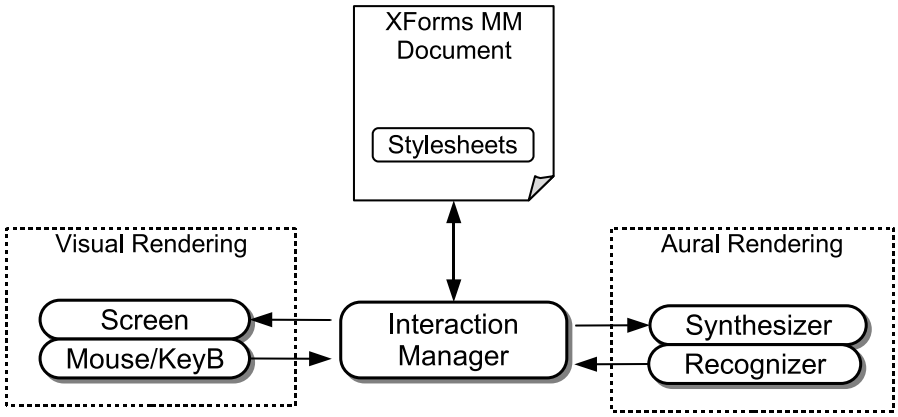

Figure 4.5: Conceptual model of the XFormsMM approach.

In order to provide an easy-to-author and maintainable multimodal user interface language, the semantic level must be raised. Raman [98] has listed several features in XForms, which are useful in creating multimodal UIs. XForms form controls, grouping constructs and event types are abstract and generic. Therefore, it suits describing multimodal user interfaces.

The Publication [P9] proposes a novel model, called XFormsMM, which includes XForms 1.0 combined with modality-dependent stylesheets and a multimodal interaction manager. The model separates modality-independent parts from the modality-dependent parts, thus automatically providing most of the user interface to all modalities. The model allows ad-hoc modality changes, so that the user can decide, which modalities to use and when.

The solutions to the requirements are listed in Table 4.4. The main concepts are the Interaction Manager and the modality-specific renderings (e.g., visual and aural), as depicted in Fig. 4.5. The interaction manager handles the input and output to and from the rendering subsystems. It also implements the sequential, simultaneous, and composite input, while the latter two are supported only partially through modality-dependent stylesheets. The main idea is that visual navigation can be performed as usually - with keyboard or mouse, while the interaction manager includes a aural focus manager, which keeps a aural focus point, and allows the user to navigate the user interface to sibling objects or up and down the hierarchy (which is induced from the document structure). It also synchronizes the aural and visual focus point in the case where the same element has both aural and visual representations.

The Publication [P9] also compares these solutions to SALT and X+V, and concludes that ease-of-authoring is much better in XFormsMM. For instance, a simple multimodal short message sending application, which was originally written in X+V[6] (9.3 KB), was only one fourth in size when written in XFormsMM (2.3 KB), although the XFormsMM version had additional features. It should be noted though, that the size of code does not automatically correlate with author productivity.

[6] X+V-based SMS sending, available at:
http://www-128.ibm.com/developerworks/wireless/library/wi-send.html.

Table 4.4: Solutions for multimodal requirements [P9].

Requirement	Solution by XFormsMM
1. General	
Supplementary interaction	Most interactions available to all modalities **automatically**.
Complementary interaction	**Modality-specific CSS properties** allow the UI author to provide some interactions only to some modalities.
Modality synchronization	Automatic, but only **at field level**.
Multilingual	**@ref attribute** in labels, help, hint, and alert.
Accessibility	**High semantic level** and form-control- and group-associated labels, help, hint, and alert.
Ease-of-authoring	Easy-to-author, since main interactions in all modalities are **simultaneously authored**.
2. Input modality	
Input processing, Semantics	**XML Schema datatypes**, and **form control types** provide semantics of input for different input modalities.
Sequential, simultaneous, composite	The interaction manager uses **modality-specific CSS** to provide some level of support of these.
Coordinated constraints	All modalities share the **same model** and instance data, so all constraints are shared.
3. Output media	Different output media is supported through the use of **modality-specific stylesheets**.
4. Architecture	
Separation of data model, presentation, and application logic	**XForms separates data model and presentation**. The application logic is usually handled in a server-side process, and then it's well separated as well.

4.4 Security through Digital Signatures

Security requirements of web applications can be divided into two main categories, secure communications and non-repudiation. Secure communications are already provided in the web by Secure Sockets Layer (SSL), and thus the focus of the Publication [P7] is non-repudiation, which is needed, e.g., in e-commerce or public services (e.g., tax report forms, etc.). Typically this is achieved with digital signatures.

An important, but often overlooked, property of a digital signing application is the capability to express the signature over everything that was represented to the user. This principle is called *What You See Is What You Sign (WYSIWYS)*. To accomplish this, it is normally necessary to secure as exactly as practical the information that was presented to that user [12]. This can be done by literally signing what was presented, such as the screen images shown to a user. However, this may result in data, which is difficult for subsequent software to manipulate. Instead, one can sign the data along with whatever filters, style sheets, client

profile, or other information that affects its presentation [6]. This, and other requirements for secure web forms are derived in Publication [P7]:

Requirements: Digital Signatures

1. *Signature security.* Technical requirements for secure signatures.
   (a) *Client-side.* The signature must be generated client-side so that the user can check the signature validity before submitting. Also, support for signing with secure smart card must be supported.
   (b) *Trusted security algorithms.* The signature must be generated using common, trusted, algorithms for maximum security.
   (c) *Signed form reconstruction.* It must be possible to reconstruct the signed form in case of dispute.
2. *Signature coverage.* The signature must cover all resources, which were used to present the form to the user (WYSIWYS).
3. *Complex signature support.* Support for complex signing scenarios.
   (a) *Partial signature.* Support signing only part of the form.
   (b) *Multiple signatures.* Support multiple signatures within one form.
4. *Form language integration.*
   (a) *Ease of authoring.* Provide as easy syntax as possible for authors.
   (b) *Ease of implementation.* Use of off-the-shelf libraries should be possible.
   (c) *Modality and host language independence.* The design should be independent of modality and host language.

Previous work in the field includes, e.g., Extensible Forms Description Language (XFDL) [13]. In XFDL, all of the information related to the form is included in a single XML file, including form definition, styling information, form data, and even attachments (in binary encoding) and the signature is created over this single file. There is also research on specific algorithms on determining whether unsigned areas are visually overlapping with signed areas [19]. This is required, if signatures over a partial form are allowed. Unfortunately, this approach relies on absolute positioning layout model, and thus it cannot easily be generalized, e.g., to XHTML+CSS, which supports a mixture of layout models.

The author has proposed a novel scheme for extending the web forms language, XForms, with secure client-side digital signatures in the Publication [P7]. The scheme uses XML Signatures [6], which provides the necessary framework for encoding, serializing, and transmitting signatures. A typical web form references multiple resources, and of them, including

Table 4.5: Solutions for signature requirements.

Requirement	Solution
1. Security	
1a) Client-side	**The signature process** is run client-side. No private keys are transferred to the server.
1b) Trusted security algorithms	The signature process follows **XML Signatures' core generation** rules [6], and uses it's default secure signature algorithms.
1c) Signed form reconstruction	The signature process is designed in such a way that it is possible to **reconstruct the form** in the way it was presented to the user at the moment of signing.
2. Signature coverage	The signature is created **over all resources** that were used to prepare the presentation for the user: The host document, all referenced URLs separately: images, objects, Applets, stylesheets, scripts, XForms external instances, xinclude, xlink, XSLT, etc.
3. Complex	
3a) Partial signature	The signature process is always done to the complete form, including all resources. Thus filtering for partial signatures must be **done at the server,** or with XForms dynamic UI.
3b) Multiple signatures	**Overlapping signatures** are supported, for separate multiple signatures, server-side process is needed for filtering.
4. Integration	
4a) Ease of authoring	The **sign** element is an XML Events handler, which creates a signature over everything that was presented to the user, and places it to the instance data at node referenced by the attribute @ref. The event **signature-done** is a DOM event that signals that signing is successfully executed. Target is the sign element that generated the signature.
4b) Ease of implementation	Only standard XML Signature operations are used.
4c) Host language independence	Since all resources are included in the signature, it does not matter, which host language and modality was used.

client-side default stylesheets need to be included within the signature. XML language module called *sign* and a related processing model for both creating and validating a client-side digital signature over a web form. Table 4.5 summarizes these and other solutions to the requirements.

4.5 Extensibility with XBL Custom Controls

The criteria defined in Section 2.6 include *extensibility*, which means that the language should not limit the range of possible user interfaces and interaction styles, and it should be possible to change parts of the user interface to suit better specific usage context or device. For example, consider a country selection, which is typical for internationalized web

applications. In a device, with better capabilities, this could be shown as an interactive map, which could be zoomed, while in a smaller device, it could be presented as a set of drop-down menus (cf. Figure 4.6). Another aspect is new kinds of interaction styles. For instance, it might be desired to use direct manipulation interaction style [103] in some usage context.

Requirements: Extensibility and Alternative Content

1. *Custom controls.* Replace parts of the UI, based on the usage context, with interactive widgets defined by the UI author or the device manufacturer.
2. *Re-use.* Allow the UI author to define new, re-usable interactors.
3. *New interaction styles.* Allow the UI author to use new interaction styles for custom controls.

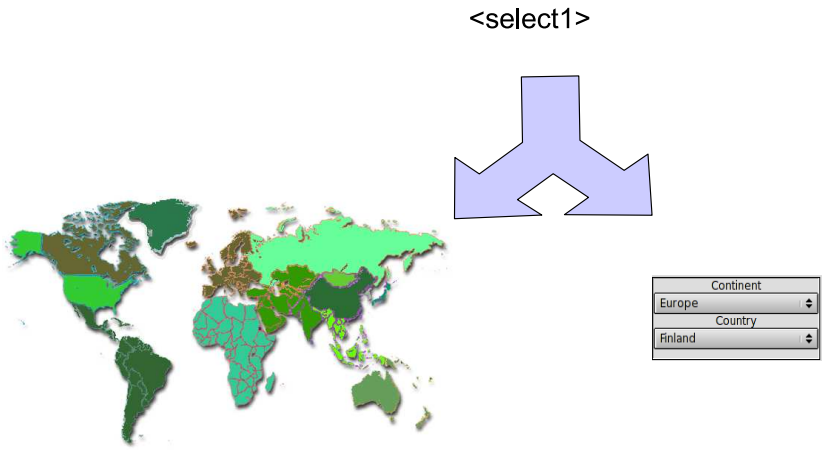

Figure 4.6: A country selection can be shown differently in different platforms using custom controls, without losing the semantics of the selection.

In order to fullfill the requirements, this Thesis proposes to solve both UI author -defined alternative content and extensibility with using **XML Binding Language (XBL)** [58], along with compound documents by inclusion (cf. Section 4.2) of XHTML+CSS and SVG, combined with well-encapsulated ECMAScript logic. Originally this idea was proposed by Birbeck for formsPlayer, which was the first implementation to support XBL with XForms[7][11]. Currently this is also supported in other XForms processors, including Mozilla XForms[8] and X-Smiles. These software distributions include several examples on how to create re-usable widgets.

The solution is summarized in Table 4.6. The main benefits of this approach are re-use and encapsulation. Each of the widgets build with XBL can be re-used as such in any other

[7]Mark Birbeck, A Standards-based virtual machine, W3C Web Applications Workshop 2004, Available at http://www.w3.org/2004/04/webapps-cdf-ws/papers/webapps-workshop-standards-based-vm.pdf. Accessed Nov 2006.

[8]Mozilla Developer Center, XForms:Custom Controls, Web Page, Available at: http://developer.mozilla.org/en/docs/XForms:Custom_Controls. Accessed Nov 2006.

user interface. Also, the way how XBL separates the generated content from the host DOM means that the procedural scripting is well encapsulated and easy to maintain. The use of SVG allows a way to provide new interaction styles, such as direct manipulation. Also, since XBL binding is applied to XForms form controls, a fall-back is to not use the binding at all, but rather use the normal XForms Form Control.

Table 4.6: Solutions for extensibility requirements.

Requirement	Solution
1. Custom Controls	XBL is used to provide custom control definitions, which can be bound to different form controls.
2. Re-use	XBL encapsulates the custom control definition (consisting of XHTML/SVG + ECMAScript) into a re-usable component.
3. New interaction styles	SVG allows the vector graphics implementation of interactors, which use new interaction styles.

4.6 Adaptation

As best demonstrated by the modality-independence (cf. Section 4.3), the proposed model allows automatic client-side adaptation of the user interface to some extent. It is sometimes necessary to do further adaptation. The main approaches to adaptation were discussed in Section 2.5.

The proposed approach for adaptation for different contexts (usage and device), is the use of single UI definition, along with context-dependent stylesheets, which are selected by using *CSS Media Queries* [67]. The method falls in the *Unique portable code* / *generic client* category (cf. Section 2.5). It is also a single-authoring method with author defined rules for adaptation. It enables fine grained control over the four main aspects of adaptation:

Spatial. Selecting the set of CSS stylesheets or fractions of stylesheets, which control the spatial (or aural) arrangement.

Temporal. Selecting the set of timesheets, which control the temporal arrangement.

Interaction. Selecting parts of user interface to be presented through CSS properties. Selecting the XBL bindings for context-specific custom controls.

Media. Selecting the media objects, or changing their display size through CSS properties.

The proposed solution, with XForms as the interaction definition language, fits the Cameleon reference framework (cf. Section 2.5) in the following way: XForms mainly lies in the *Concrete UI* layer, but its' data model, abstract grouping and form controls are modality-independent, which means that it covers some parts of the *Abstract UI* as well. The modality-dependent stylesheets can be considered to be in the *Final UI* layer. The *Tasks & concepts* layer is not covered by the proposed model.

A model-based approach could be used to provide higher-level semantics (e.g., the Task & concepts layer), while providing lower-level templates using the proposed model. Also, an interesting approach for dynamically processing an XForms-based UI description to fit to smaller screens is described by Book et al [16]. Their approach is promising, although it is not clear wheather it could support the more complex UI patterns, such as dynamic UI bindings, filtering, or master-detail (and to transform between patterns based on the context).

4.7 Summary of the Proposal

The requirements for the UIDL and the proposed solution is summarized in Table 4.7. The proposal is a compound document profile consisting of many XML languages (and one non-XML language, CSS) specified, or being specified, by W3C. Typically, the languages are unchanged, but in some cases they are extended (XForms Tree proposal), or only few modules of the language in included (SMIL). Sometimes the integration of languages has required the Author the add new elements to the language (e.g., the Timesheets proposal).

Table 4.7: The requirements and proposed solutions for the UI description language profile. The profile is mostly composed of W3C-standardized, or to-be-standardized languages.

Requirement	**Proposed Solution**	**Chapt.**
Interaction	XForms 1.1 + Tree extension for nested structures	4.1
Visual Style	CSS 2.1 visual properties.	4.2
Spatial Layout	CSS 2.1 layout model and properties.	4.2
Continuous media	XHTML 1.1 object element provides support for video and audio elements.	4.2
Discrete Media	XHTML 1.1 and SVG 1.1 for vector graphics combined with XHTML both by reference and inclusion.	4.2
Temporal Dimension and Animations	SMIL Timing And Synchronization and Animations modules [57] integrated with XHTML and SVG, used with the timesheet element, as suggested by [RP5].	4.2
Modality-Independence	XFormsMM (XForms 1.0+Visual and Aural CSS)	4.3
Security	HTTPS for secure transmission. XML Signature integrated with XForms for digital signatures.	4.4
Extensibility	XBL and ECMAScript for re-usable custom interactors, SVG for new interaction styles.	4.5
Adaptation	CSS Media Queries combined with context-dependent CSS stylesheets and timesheets for adaptation of spatial, temporal, interaction and media arrangement.	4.6

5 PROOF OF CONCEPT IMPLEMENTATION

The results of this Thesis have been validated by proof-of-concept implementations. These implementations have, almost exclusively, been built into the open-source XML browser, X-Smiles. The author has been one of the main designers and implementers of the X-Smiles browser.

5.1 Requirements

The main requirement for the X-Smiles browser was to support the proposed language profile. The other main requirements was device independence, which is composed of Portability and Adaptability. The requirements can be further divided into the following (extended from [P6]):

1. *Multimedia Language Support:* support for the layered model of XML languages, as defined in Section 4, including XML, DOM, ECMAScript, XHTML, CSS, CSS Media Queries, SVG, XForms, XFormsMM, Timesheets, XML Signatures, and XBL.

2. *Portability*

   (a) *Code Portability*: the user agent has to be implemented in such a way that it can be run in different platforms.

   (b) *Toolkit:* each graphics system (e.g., Symbian and Gnome) provides a different toolkit. Thus, the user agent has to be able to support any toolkit.

   (c) *Media Providers*: each platform uses different providers to render the media. For example, it provides native methods to play video or audio. Thus, the user agent has to be able to support any media providers available in the platform.

3. *Adaptability*

   (a) *Input Mechanism*: the way the user interacts with the system depends on the device at hand (e.g., digital television receivers use remote control). Thus, the user agent should provide alternative methods to navigate/activate content.

   (b) *User Interface Adaptation*: the characteristics of the graphical display (e.g., size) are particular to the device. Thus, the user interface of the *browser application* should be adapted (e.g., television display is normally viewed from the distance, so the fonts should be at least 16 points). Note that this requirement does not mean adaptation of the content, which is handled in Section 4.6.

5.2 X-Smiles Architecture

The architecture of the X-Smiles user agent is depicted in Figure 5.1 (extended from the Publication [P1]). The XML Stream is handled by the XML processing layer. It includes an XML Parser and an XSLT processor, which generate a stream of SAX events, which is forwarded to the browser core layer. The core layer has DOM implementation and the XML broker module, which are presented in Publication [P2]. The MLFC layer includes DOM element implementations for all supported namespaces, which are registered to the XML broker. XML broker uses the DOM implementation to create a DOM tree, which contains specialized element implementations for each namespace in the document. Binary content (audio, video, images, and plain text) is handled by the content manager, which is usually summoned by an MLFC (e.g., HTML img element), or directly by the browser GUI (e.g., when the user navigates directly to an image URL).

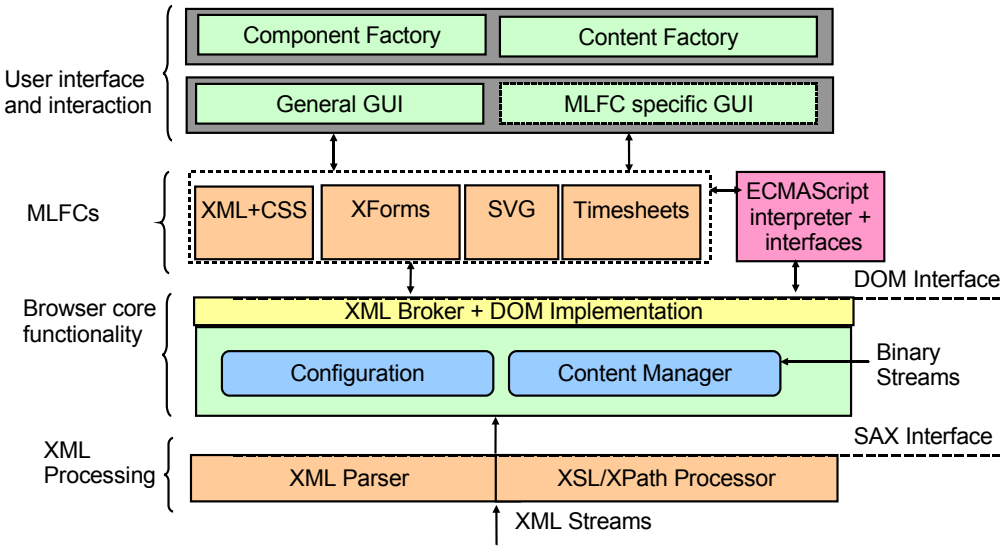

Figure 5.1: Architecture of the X-Smiles user agent (extended from [P1]).

On top of the Figure 5.1 lies the use interface layer, containing the general browser GUI along with MLFC specific GUIs (e.g., stop and play buttons for timed content). The ECMAScript interpreter[1] is used by the language implementations, and it accesses the document through the DOM interface.

5.3 Portability

Portability is achieved using two main approaches: Java as the programming language (code portability) and a device-independent abstract toolkit layer for media objects and the UI components (UI portability).

[1]Mozilla Rhino implementation is used in X-Smiles. Available at: *http://www.mozilla.org/rhino/*. Accessed Nov 2006.

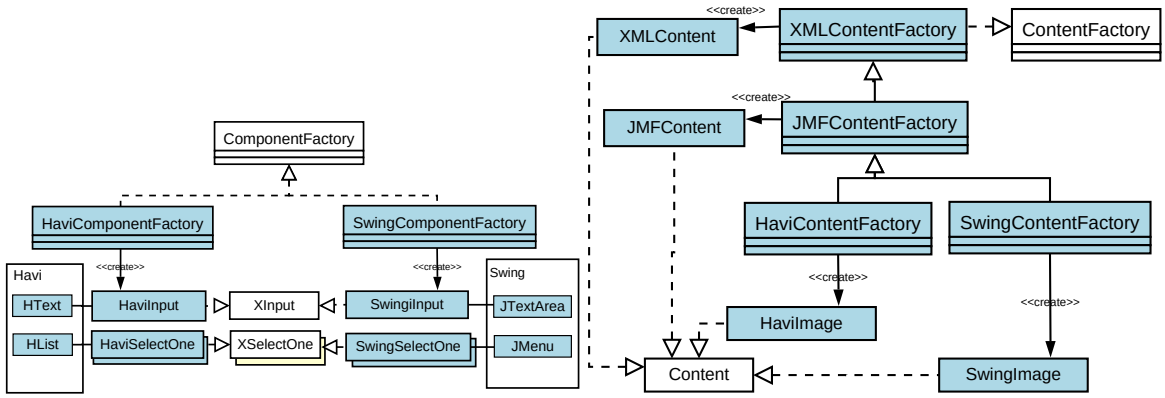

Figure 5.2: (a) Component [P6] and (b) Content Factories.

Content and component factories handle the creation of different types of widgets and media providers, as depicted in Figure 5.2. Media providers differ from widgets in three ways: they do not have detailed events for user interaction, they consume a media stream, and they provide synchronization primitives (start, stop, and pause commands and corresponding events). The content is identified by its content type (i.e., Multipurpose Internet Mail Extensions, MIME type), and it is possible for different factory implementations to support different set of content types, depending on the lower level capabilities.

Note that all XML language implementations have to be implemented using this layer, so that they access toolkits and media libraries only through the corresponding factory. The Publication [P5] describes the architecture of the XForms processor, focusing on the device independence.

5.4 Adaptability

As described in the Publication [P6], browser UI adaptation was solved by implementing a adaptable browser UI, which contains a core UI description and derived UI descriptions for possible device categories. The device-independent core UI is build so that there are generic actions, which can be mapped to different device-specific toolkit widgets or actions, such as menu items. This way it is possible to use the same GUI in different environments, thus making maintenance much easier, compared to an earlier approach of writing a new GUI for each new device.

As a proof-of-concept, X-Smiles has been run in 3 embedded environments: a PDA, a digital TV receiver, and a smartphone (cf. Figure 5.3).

5.5 XForms Implementation

One of the main contributions of this Thesis is a complete browser implementation of the XForms 1.0 Recommendation. This implementation by the Author was the first complete

(a) (b)

(c)

Figure 5.3: X-Smiles running in a (a) handheld device, (b) TV receiver, and c) smartphone

implementation of XForms, and one of the three implementations, that allowed XForms to become a W3C Recommendation. The others were xPort FormsPlayer[2], and Novell XForms Preview[3]. The early implementation is discussed in the Publication [P1], while an updated description can be found in the Publication [P5].

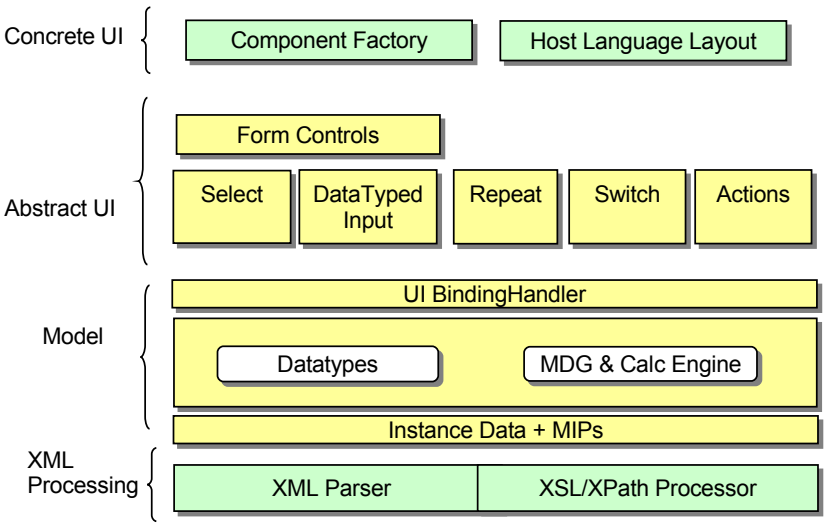

Figure 5.4: Architecture of the XForms processor.

The architecture of the XForms processor is described in Publication [P5]. The Figure 5.4 depicts the architecture, which contains four layers. The XML processing layer contains an XML Parser, and an XPath processor. The model layer contains the instance data, MIP, datatypes, and the Master Dependency Graph (MDG). Also, the UIBindingHandler is in the model layer. The abstract UI layer contains implementations of the specific XForms UI elements, such as input, select, repeat, and switch. Also, datatype-specific form controls are located here. On the top, there is the concrete UI layer, where the ComponentFactory and the host language layout functions are located.

The heart of the XForms processor is the model implementation, which is described in Publication [P5], and whose architecture is depicted in Figure 5.5. On the bottom lies the XML processing layers with XML parser, DOM, XPath, serialization, and schema implementations. In the middle is the core and abstract UI layers. The topmost layer in the Figure 5.5 is the UI Binding Handler, whose responsibility is to track dynamic UI dependencies, and dispatch events, when dependencies change (and thus the UI bindings become possibly dirty).

The responsibilities, and the solutions, of the model implementation are the following:

Parsing and storing the instance data is implemented using the Xerces XML Parser, which is used to parse the stream into a DOM L3 representation, including PSVI infoset additions for XML Schema.

[2]xPort FormsPlayer, Software, available at *http://www.formsplayer.com/*, Oct 2006.
[3]Novell XForms Preview, Software, not available online.

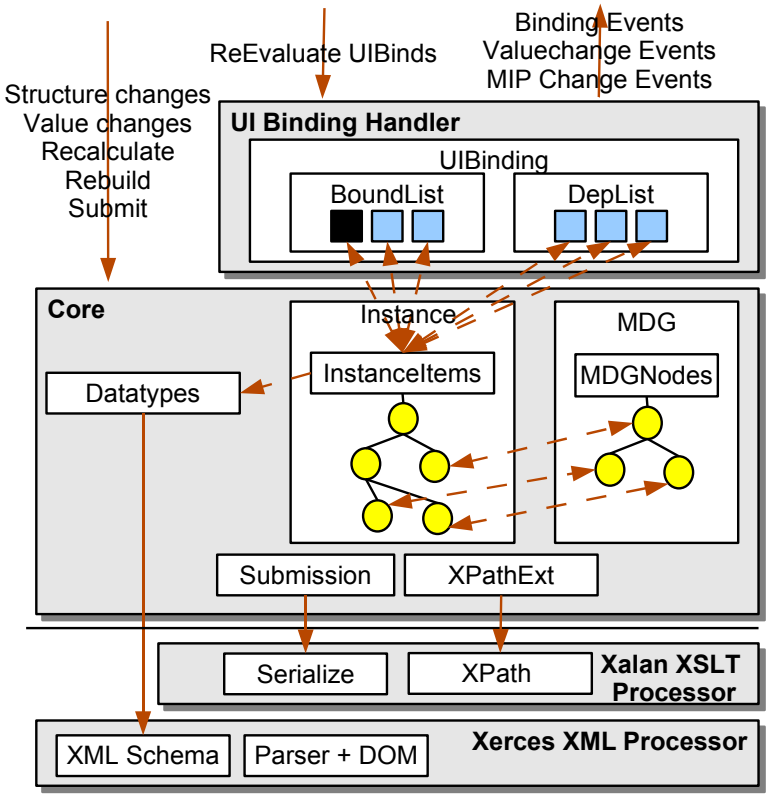

Figure 5.5: The XForms model architecture.

Master dependency graph (MDG) is implemented in a separate module, which contains the algorithms for dependency graph initialization and recalculation with dirty nodes. The publication [P3] describes the effort to add efficient computation engine into the XForms language. At the time of publication, the UI author had to declare the calculation order of all the dependencies in the user interface, and the proposed calculation engine adds spreadsheet-type of calculation order resolving into XForms. That algorithm was later accepted as such into the XForms Recommendation. The engine relies on well-known graph algorithms (mainly topological sort [62]) that optimize the search when looking for the calculation order of the XForms constraints.

The model item properties (MIP) and their inheritance is implemented in a set of classes derived from a MIP base class.

Datatypes are implemented in a separate class structure, which uses Xerces datatypes implementations, whereever possible.

Submission is a module containing the different submission serializations, methods, and protocols. It uses the Xalan serializer for XML serializing.

XPath extension functions are implemented once independent of the used XPath engine. Each integrated XPath engine (Xalan, Jaxen) is then extended so that the extension functions are in the scope.

Dynamic UI dependencies Dynamic UI dependencies are contained in a module, which tracks any dependency change, and informs the UI binding owners of possibly dirty UI bindings.

The UI layer of the processor is composed of *form controls*, *repeat* and *switch* processors, and the implementation of *actions*. These are described in detail in Publications [P4,P5, RP2].

5.6 Multimodal XForms

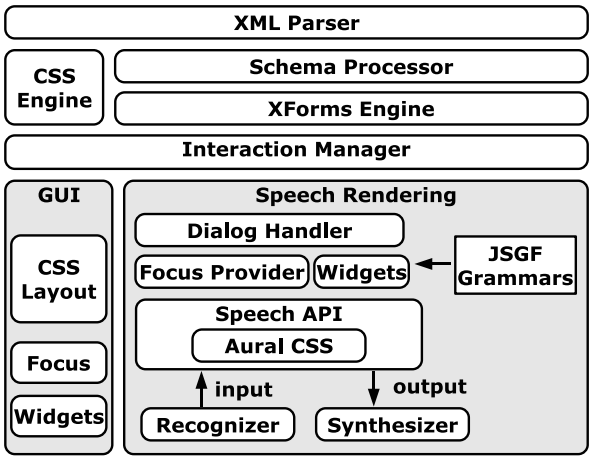

Figure 5.6: The components of the XFormsMM implementation in X-Smiles.

The XFormsMM (XForms Multimodal) framework was proposed in the Publication [P9] (cf. Section 4.3). The framework was designed by the Author and implemented in X-Smiles by the Author and Mikko Pohja. The implementation (cf. Figure 5.6) consists of an interaction manager, which implements the algorithms proposed in [P9], and the speech rendering of XForms language. The speech rendering handles the speech input and output between the processor and the user.

Sphinx-4[4] was used as the speech recognition engine and FreeTTS[5] as a speech synthesis engine, while the Speech API, designed by the Author, allows an easy way to plug in different ASR and TTS engines in the future. Both of the used libraries are open-source cross-platform Java libraries. Specialized JSGF grammars were written for all typical XML Schema basetypes in XForms. The grammars are interpreted by the Sphinx-4 library.

5.7 Integrating Digital Signatures to XForms

The author has implemented the current complete implementation with the processing described in the Publication [P7] (cf. Section 4.4). It is based on an earlier experimental

[4]Sphinx-4, Software, Available at *http://cmusphinx.sourceforge.net/sphinx4/*. Accessed Nov 2006.
[5]FreeTTS, Software, Available at *http://freetts.sourceforge.net/docs/index.php*. Accessed Nov 2006.

implementation, which was done in co-operation with Heng Guo [53].

The signature processor was integrated with the XForms processor in the X-Smiles browser. The sign element and it's namespace were created as a new MLFC. The XML signature is created using the *Apache XML Security 1.1 for Java* library[6]. This library is a complete XML Signature implementation. It is also responsible for creation of the external references in the signature.

The signature MLFC uses well-defined interfaces for inter-component communication. In the integration of XML Signature and XForms processors, DOM interfaces were utilized as much as possible. The signature element is part of the UI DOM tree, so it can naturally access the tree with the XML and XForms DOM access. The user agent must provide an interface to access the list of resources, which were loaded for the form being signed. Some of these resources, such as images, can be referenced via an absolute URL, while others are local to the processor. Both must be provided through this interface so that they can be included within the signature.

5.8 Custom Controls with XBL

The XForms custom controls framework described in Section 4.5 was implemented in X-Smiles by the Author. The implementation is a subset of XBL. A more complete implementation of the same idea can be found in formsPlayer[7] and the Mozilla XForms implementation[8]. The author's partial implementation follows the Mozilla implementation.

The Figure 5.7 shows a screenshot of X-Smiles running a form for selecting a city in Finland. The city can be selected either in a normal combo-box or by clicking the city in the map. There are also different versions of the city name output: one without custom skin, and one with SVG custom control. All controls are automatically synchronized so that the current value is always presented in each of them.

5.9 CSS Layout Engine for Compound Documents

The CSS Layout engine has been co-designed by the Author and Mikko Pohja, and mostly implemented by Mikko Pohja (the Author has done some implementation work related to integration of interactive components, and synchronization between incremental re-layouts and scripting, etc.). The layout engine is described more in detail in [92]. The basic operation is incremental layout of an XML document based on CSS properties. This is depicted in Figure 5.8.

The implementation is based on a *single, shared DOM model*, where the different language implementations share the same object model (along with the separate *layout or view*

[6]XML Security for Java and C++, Software, Apache Foundation, Software. Available at *http://xml.apache.org/security/*. Accessed Nov 2006.

[7]formsPlayer, software, x-Port, available at *http://www.formsplayer.com/*. Accessed Nov 2006.

[8]Mozilla XForms Project, Software. Available at *http://www.mozilla.org/projects/xforms/*. Accessed Nov 2006.

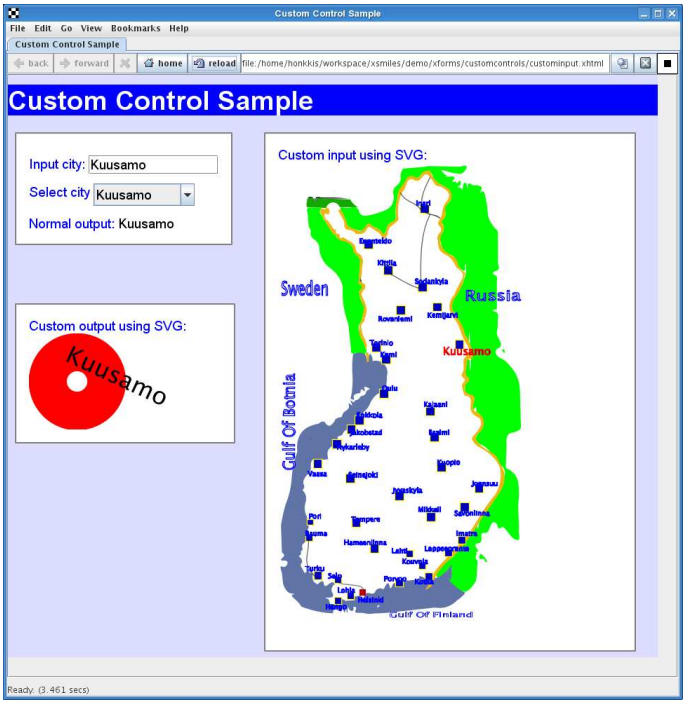

Figure 5.7: Different implementations of the city selection using XBL+SVG custom controls in X-Smiles.

tree), which is generated at runtime by the XML parser. The author's recommend this kind of approach because of the number of different properties (such as CSS properties and DOM events) that must flow through the document model dynamically based on window size and DOM changes.

The author has integrated SVG into the compound documents layout engine. The integration was done by modifying the CSIRO SVG library. The biggest modification was to move all functionality from specialized SVG document implementation into the SVG root element implementation. Then, the root element implementation was extended from X-Smiles default element implementation in order to be able to place it within a XHTML document. This integration allows unlimited number of SVG root elements to be placed within the XHTML document, as shown for instance in Figure 5.7. The CSS properties inheritance from XHTML to SVG was implemented together by the Author and Alessandro Cogliati in the XForms 1.0alpha1 release.

5.10 Timesheets

The implementation of the Timesheets idea, presented in the Publication [RP5], was co-designed for the X-Smiles browser by the Author, Teppo Jalava, and Mikko Pohja. Teppo Jalava did the main implementation work.

Since Timesheets only describes the temporal dimension of the document, it was integrated with a CSS layout implementation in X-Smiles. The integration works by affecting

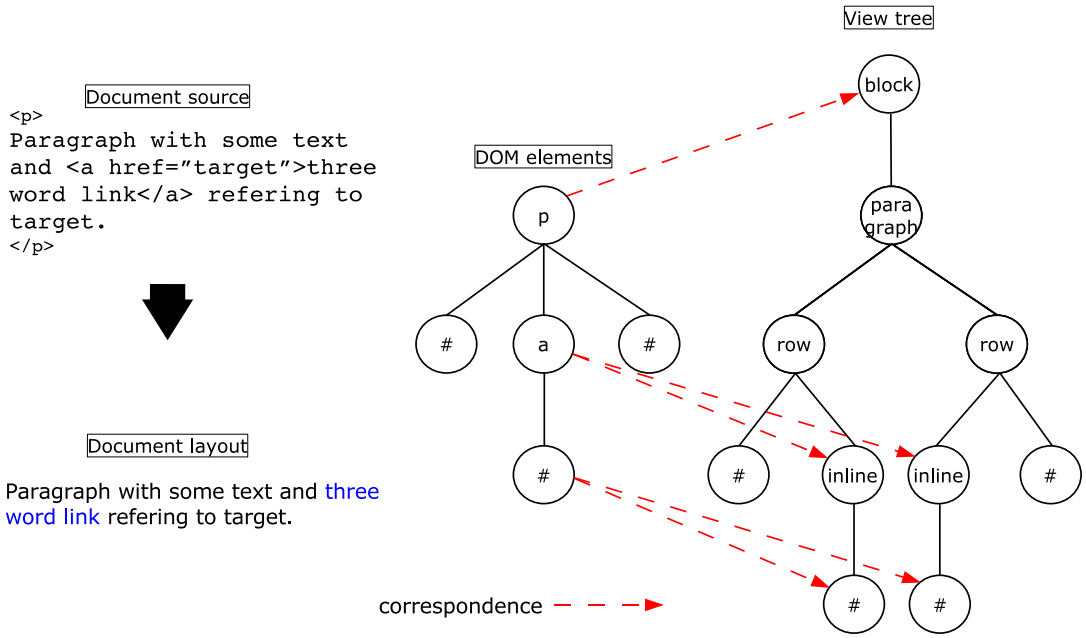

Figure 5.8: CSS Text Layout example [92]

the CSS properties of media elements based on the current time point. For instance, the CSS property *display* controls whether the element is displayed or not. The timesheets processor sets the CSS pseudo-class *timed-inactive* for the media element when the element should not be visible based on the timesheet. Then, with a default rule for the pseudo-class, the CSS property display is set to "none" to the non-visible media element. The pseudo-class overrides element's original display property and the element becomes invisible. When the media element should become visible, the implementation removes the pseudo-class from the element, thus restoring the original value.

The default style sheet should contain a declaration with the pseudo-class called timed-inactive that is used to control the visibility of the media elements:

```
:timed-inactive { display: none ! important;}
```

Additionally, the language is extended by defining a class attribute for the item element. It can be used to change style properties of the elements from the timesheet. For example, the UI author could specify the red background of an element to be changed to blue at some time. This is done by setting the value of class attribute to reference a CSS pseudo-class that is used in the style sheet of the document. The UI author doesn't have to add the same document element with the same content multiple times with different style properties, but instead only define a pseudo-class for each property change and control them from the timesheet.

Future work includes implementing SMIL Animations module, and integrating Timesheets to the SVG implementation.

5.11 Summary of the Implementation

The requirements for the UIDL and the proposed solution are summarized in Table 5.1. The largest part of the implementation handles the first requirement: Multimedia Language Support.

Table 5.1: The requirements and proposed solutions for the implementation.

Requirement	**Proposed Solution**	**Chapter**
Multimedia Language Support	Implementation of the proposed model (cf. Section 4.7) in an XML user agent X-Smiles.	5.5 - 5.10
Portability	Using Java as the programming language allows portability to different devices. Design and implementation of a virtual toolkit, which abstracts the actual toolkit and media libraries in the device.	5.3
Adaptability	The browser GUI is designed so that it can easily be subclassed for different devices while still having the common functionality (such as navigation) in the main class. The different input mechanisms are supported by using the native toolkit in each device.	5.4

6 CONCLUSIONS

This Thesis has proposed a set of high-level user interaction languages for implementing device-independent web user interaction. The motivation has come from new requirements for web applications: *higher interaction*, *adaptation* for mobile and multimodal usage, and rich *multimedia* content.

6.1 Validation

The validation of the results in this Thesis is twofold. First, *theoretical validation* is performed by comparing the proposal against similar approaches using the general and web-specific criteria, defined in Sections 2.6 and 2.7 respectively, and the more detailed requirements in each Section of Chapter 4. Second, *empirical validation* is done by the proof-of-concept implementation (cf. Chapter 5) and the implemented use cases from various application scenarios defined in Section 2.2

Regarding the *theoretical validation*, current web UI languages were reviewed in Chapter 3. The matrix, which compared the approaches against the criteria is extended here with the proposed solution of this Thesis in Table 6.1. Note that this table is based on author's understanding of the state of the art, and is therefore somewhat subjective. Also, Tables 6.2 and 6.3 describe the fulfillment of each of the criteria by the proposed solution. The Thesis' proposal covers quite well the different criteria, having the biggest weakness in *Ease of implementation*. This stems from the facts that the proposal includes a lot of different XML languages, whose implementations need to interoperate on different levels. For instance, SVG and XForms must integrate to the CSS and events subsystem of the host XHTML document.

Note that the proposed solution has combined many of the compared languages into a larger whole, which makes this comparison somewhat unfair. Thus, an additional language profile (Ajax+SVG+Web Forms 2.0) was added to Table 6.1. Compared to Ajax+SVG+Web Forms 2.0, the Thesis' proposal is better at *device and modality independence*, *ease of authoring*, and *authoring tool interoperability* mainly because of higher abstraction level. On the other hand, *ease of implementation* is better in Ajax+SVG+Web Forms 2.0 (and some browsers already implement this profile, at least partially[1]).

The *empirical validation* has been done first by the proof-of-concept implementation (cf. Section 5). It acts as the open-source reference implementation of the proposed model, and also can be used to assess the implementation implications. Another kind of empirical validation has been the implementation of use cases from various application scenarios. Please

[1]Mozilla Firefox 2.0 and Opera 9.0 both support Ajax and subset of SVG 1.1 natively, and Opera 9.0 supports Web Forms 2.0 as well.

Table 6.1: Comparison of the proposal to the other tools reviewed in this Thesis. Empty cell means less than average, '+' means average, and '++' means better than average.

	General Criteria							Web Specific						
	Expressivity			Authoring										
Tool	Visual Presentation	Interaction	Multimedia / Synchron.	Ease of learning	Ease of authoring	Extensibility	Authoring tool interop.	Device Independence	Modality Independence	Low latency	Web Integration	Accessiblity	Security	Ease of implementation
Applets*)	++	++	+			++				++			++	
WebTK *)	++	++	+		+	++				++	+		+	+
DHTML*)	+	+	+			++		+		+	++		+	+
Web Forms*)	+	++	+	+	+	++	+	+	+	+	++	++	+	+
Ajax*)	+	++	+			++		+		++	++	+	+	+
XUL*)	+	++				+				++		+	+	+
SVG*)	++	+	+			+	+			++	+		+	+
AjaxSVG WForms*)	++	++	++			++	+	+	+	++	++	++	+	+
XHTML	+			++	++		++	+	+		++	+	+	++
SMIL	+		++	++	++		++	+	+	+	+	+	+	++
XForms	+	++		+	++		++	++	+	++	++	++	+	+
Proposal	++	++	++	+	+	+	++	++	++	++	++	++	++	

refer to Section 2.2 and the publications referenced there for more details.

6.2 Main Contributions

A contribution of this thesis is a taxonomy of web user interface tools, based on authoring style, has been proposed. The first class, traditional user interface tools, namely toolkits, are *procedural*, which require UI authors to be experienced programmers. The second class is *declarative* UI languages. The third class is *hybrid* UI tools, which is a mixture of declarative and procedural (e.g., DHTML + scripting).

Another contribution is a definition of criteria for user interaction tools within the application scope. Current tools are also reviewed against these criteria in this Thesis.

The main contribution of this Thesis is a proposal for combined set of declarative languages to describe device- and modality-independent user interfaces. Additionally, one of the contributions has been the involvement in W3C XForms Recommendation process, and

Table 6.2: General criteria and their fulfillment by the proposed model.

General Criteria	Fulfillment by the Proposed Solution	Ch.
Expressivity Visual Presentation	The expressivity of visual presentation is good in the proposed model, since different layout models and primitives are supported through CSS 2.1 and SVG.	4.2
Expressivity Interaction	This criterion has been the focal point of this work. XForms 1.1 has been designed to address most of the interaction requirements of the applications within the applications scope.	4.1
Expressivity Multimedia and synchronization	This criterion is well fullfilled by the inclusion of few core SMIL modules using the Timesheets technique.	4.2
Ease of learning	Not proven. This criterion depends on the target authors, and should be empirically proven. The Thesis claims that at least for non-programmers declarative languages are easier to learn compared to procedural, and thus the main core of the proposed model uses declarative languages, such as XForms, and CSS.	
Ease of authoring	Not proven. Similar to the previous criterion, this depends also on the target authors. The claim of this Thesis is that declarative languages are easier to author for non-programmers (cf. ease-of-use of tools [80]).	
Extensibility	Few methods of extensibility is supported by the model. First, implementation-specific extensibility allows the addition of new interactors based on datatypes. Also other kinds of functionality can be added to implementations. A more generic extensibility mechanism is provided by custom controls. In this criterion, procedural approaches, including AJAX are better, though. Another aspect of this criterion is *re-use*, which is provided by the proposed solution by allowing to encapsulate custom interactors, written in ECMAScript and XHTML or SVG) by using XBL. The re-use of instance data, and thus e.g., labels, help, hint, and alerts is allowed as well.	4.5
Authoring tool interoperability	Not proven, though this Thesis claims that the use of declarative languages in the core of the model supports interoperable authoring tools.	

feedback to the XForms working group. The author has provided the first complete implementation to help standardization process. All extensions proposed in this Thesis have been proposed to the XForms Working Group to be utilized in future standardization.

The main research question was **a device and modality-independent model for high-interaction web user interfaces**. The main contributions, and how they are linked to the publications and the four subquestions of this Thesis can be summarized as follows:

Q1: Requirements for web UIDL.

- Chapter 2, Publications [P4,P6,P8,P9].
- Main contributions: Classification of web applications based on interactivity, evaluation criteria for web UIDLs.

Table 6.3: Web-specific criteria and their fulfillment by the proposed model.

Web-specific Criteria	Fulfillment by the Proposed Solution	Ch.
Latency	Latency is minimized by allowing asynchronous submissions while the user interacts with the page. This is similar to the submission model of AJAX.	3.2
Device independence	This criterion is met by having enough high level of abstraction and layering the model such that the core UI description is independent of any end-user device, and allowing author-defined rules for context-specific adaptation. This criterion is also demonstrated by running the implementation and some of the use cases in various devices, ranging from hand-held devices to TV set-top-boxes, and desktop computers.	4.6, 5.3, 5.4
Modality independence	The modality independence of the model is demonstrated by the XFormsMM proposal, and related proof-of-concept implementation.	4.3
Security	HTTPS for secure transmission. XML Signature integrated with XForms for digital signatures.	4.4
Accessibility	XFormsMM demonstrates the accessibility features of the proposed model.	4.3
Web integration	The proposal is composed of W3C standardized or to-be-standardized languages. When appropriate, the Author participated the standardization process.	4.7
Ease of implementation	This is possibly the weakest point of the proposed model, since it requires the integration of many XML languages and technologies. This criterion is still fullfilled to some extent by providing a open source proof-of-concept implementation.	5

Q2: Evaluation of current UIDLs.

- Chapter 3, Publication [P8].
- Main contributions: Classification current web UIDLs, evaluation of current web UIDLs against the evaluation criteria.

Q3: Compound document profile to fit the requirements.

- Chapter 4, Publications [P3,RP4,P4,P7,P9].
- Contribution: Providing a model for defining web interaction, based on higher semantic level user interface description. The model allows device independence, accessibility, security, and multimodality. For summary of the model, see Table 4.7.
- Contribution: Working as a participant of the W3C XForms Working group, co-specifying the XForms 1.0, XForms 1.0 Second Ed. Recommendations, and XForms 1.1 Working Draft. Since the work has been done in a group under a non-disclosure agreement, the exact contribution cannot be enumerated here.
- Contribution: Several proposals for enhancements in XForms standards, including editing recursive structures with a tree module and digital signatures.

Q4: Implementation implications.

- Chapter 5, Publications [P1,P2,P3,P5,P6,RP1].
- Contribution: This Thesis has provided an experimental web user agent, X-Smiles, whose main designer and implementer the Author has been.
- Contribution: Implementation of the XForms 1.0 Recommendation in the browser.
- Contribution: Implementations of the following browser components: media-type aware content handler framework, device-independent GUI framework, whose sole implementer the Author has been.
- Contribution:Implementations of the following browser components: speech user interface, vector graphics plugin, digital signatures, which have been partially implemented by the Author.

6.3 Benefits and Drawbacks of the Solution

Regarding to the *proposed model*, this Thesis claims that it would be beneficial to use declarative UI languages in many usage scenarios, the main advantage being the higher semantic level compared to procedural approaches. This makes possible automatic transformations of the UIs based on context of use, current device, and user's preferences. For instance, it is possible to automatically create multimodal user interfaces from a single declarative user interface definition. As shown by the multimodal scenario (cf. Section 4.3), user interfaces written with the proposed language can adapt to many different usage scenarios. Also, it should be easier to create interoperable visual authoring tools for declarative languages, compared to procedural languages. Note that the user interfaces of some classes of interactive applications, such as realtime 3D games, should still be implemented with procedural tools (e.g., C++, OpenGL, or Java Toolkits), since they require very high level of interaction and complex visual effects.

The proposed model of declarative languages is layered, so that parts of it can be used even without implementing the whole model. For instance, the timing part (Timesheets) is a separate language module, that only depends on CSS layout, and can easily be left out if not needed. Also, the security part is optional. This can be beneficial, if the whole profile proofs to be too hard to implement.

Since the proposed technologies are targeted to be implemented in the user agent, the biggest drawbacks are the *implementation implications* (cf. Tables 4.7 and 6.1). Thus, one of the biggest efforts in this Thesis has been the *proof-of-concept implementation*, X-Smiles. The main benefit of X-Smiles is its portability; it is written in pure Java, which makes its porting much easier compared to other (mainly C++) approaches. The drawback is the performance; although Java virtual machines have developed, an algorithm written in C++ will still outperform the Java version in startup time, execution speed, and memory usage.

It also remains to be seen whether the proposed set of languages gain enough momentum to force browser manufacturers to implement them. They may make application authoring easier, and user experience better (because of device independence), but require development

stakes, which many browser manufacturers are not willing or able to make. For instance, currently XForms is not yet widely in use; it lacks mainstream browser support, although, e.g., Mozilla Firefox has an installable XForms plugin. There is an evolution path available: using ECMAScript to implement the language profile. For instance, two such open-source implementations of XForms exists already to date and implementations of e.g., Timesheets should be plausible. In this way, it would be possible to use at least the main parts of the language profile already before they are widely supported natively in the browsers. The problems related to the use of ECMAScript (efficiency and bandwidth consumption) can be somewhat reduced by using caching and implementing ECMAScript Just-In-Time (JIT) compilers in browsers[2].

Declarative languages have some general drawbacks as well; their application scope is usually smaller compared to the procedural languages. When the user interface is not within the "sweet spot" of the language, it may be very difficult, if not impossible, to define it with the declarative language. This is where the strength of procedural languages is: with them, it is usually easier to define re-usable abstractions to deal with more complex UI requirements. On the other hand, within the application scope, declarative languages can be easier to learn and use, especially for non-programmers (consider SQL over procedural search of linked tables).

6.4 Future Work

In the future, it should be studied how different adaptation techniques (e.g., model-based techniques) could be used in conjugation with the proposed model. Another future work item is related to validation. It might be possible to validate the findings of this Thesis by using usability research methods applied to non-programmer authors.

Comparing different end-to-end solutions for building rich multiuser internet applications, including database connectivity and concurrency, with the proposed model, is also left as a future work.

Regarding the implementation, more work should be done to improve the X-Smiles browser. The biggest problems currently are website compatibility and processing and memory efficiency, especially for small devices. Also, the SVG, XBL, and CSS implementations are not complete. Also, the timesheets implementation should be extended to support animations and it should be integrated with vector graphics.

[2] At the time of writing Adobe donated the ECMAScript engine from Flash 9, including a JIT compiler to the Mozilla project. The new "Tamarin" engine is expected to perform much better compared to the current Firefox 2.0 ECMAScript interpreter. Available at: *http://www.mozilla.org/projects/tamarin/*, Web Page, Accessed Nov 2006.

7 SUMMARY OF PUBLICATIONS AND CONTRIBUTIONS OF THE AUTHOR

This Chapter gives a summary of the publications, which form this Thesis. Also, it describes the contributions of the Author in each paper. The work presented here is part of a larger project, where X-Smiles has been developed. Specifically, Dr. Kari Pihkala has designed and implemented the SMIL player referred to in the articles [P1], [P2], [RP4]. M.Sc. Mikko Pohja has co-designed with the Author and fully implemented the dynamic CSS layout engine referred to in [P4], [P5], and [P6]. He also participated in the design and implementation of the multimodal interaction framework in [P8]. Dr. Pablo Cesar provided the digital television prototype in [P6]. M.Sc. Alessandro Cogliati has co-designed with the Author and fully implemented the CSS properties engine used in many of the publications. Dr. John Boyer designed the computation engine referred to in Publication [P3]. Teppo Jalava and Mikko Pohja co-designed with the Author the Timesheets proposal and implementation [RP5], which was implemented mainly by Teppo Jalava.

The Publication [P2] has previously formed part of Kari Pihkala's Doctoral Thesis, and [P6] has formed part of Pablo Cesar's Thesis. Other publications have not previously formed a part of another Thesis.

Publication [P1]

This publication presents the general approach to web user interaction using the XForms language. This is an early journal paper, that demonstrates the host-language neutrality of the XForms language approach, with detailed mark-up examples and implementation details for each host language (XSL-FO, SVG, SMIL). The author has designed and implemented all of the work presented in the paper. He has also written all of the text, while getting comments and proof-reading help from Prof. Petri Vuorimaa.

Author's overall contribution to the publication: 95 %.

Publication [P2]

This article presents a framework for an XML browser capable of displaying hybrid XML documents. The framework enables the implementation of extensions for existing XML languages. As an example, SMIL, XForms, and XML Events languages are integrated. The authors Dr. Pihkala and Honkala designed and implemented the proposed framework, and wrote the text, in equal proportion.

Author's overall contribution to the publication: 50 %.

Publication [P3]

This publication presents a successful effort to change the computation engine in the then current XForms Working Draft so that the calculation order is automatically derived. The author wrote most of the text in Section 1 and 3 and implemented the system in the X-Smiles browser. He also fully implemented the use case that is referred to in the paper. Dr. John Boyer designed the algorithm used in the computation engine, and wrote the text in Sections 2 and 4.

Author's overall contribution to the publication: 50 %.

Publication [P4]

The next version (2.0) of XHTML, which was at Working Draft stage, is introduced, and most notable changes since XHTML 1.1 are described: document structure, navigation lists, and the new forms module, XForms. The author has designed and implemented the integration of XHTML and XForms, and the integration of XForms and CSS layout. He has also written half of the text.

Author's overall contribution to the publication: 50 %.

Publication [P5]

Re-design of the XForms processor, which allows it to be ported easily to many platforms. Requirements for a portable XForms processor are derived, and a design fullfilling these requirements is presented. The author has solely done the design and implementation, and written the text, while getting comments and proof-reading help from Prof. Petri Vuorimaa.

Author's overall contribution to the publication: 95 %.

Publication [P6]

This publication derives requirements for a language to develop interactive, networked multimedia applications. A corresponding XML language profile, including SMIL and XForms) is described. Also, requirements are derived for a device-independent user agent for this language profile. It is shown that a modified version of X-Smiles can serve as such user agent. Specific solutions for supporting devices with different input and output capabilities are Component and Content factories, which are described in the paper. Dr. Pablo Cesar described the porting of this profile and user agent in an digital television prototype platform, called Ubik. The author designed and implemented the factories and the device-independent GUI framework.

Author's overall contribution to the publication: 50 %.

Publication [P7]

The World Wide Web is evolving from a platform for information access into a platform for interactive services. The interaction of the services is provided by forms. Some of these services, such as banking and e-commerce, require secure, non-repudiable transactions. This

paper presents a novel scheme for extending the current web forms language, XForms, with secure client-side digital signatures, using the XML Signatures language. The requirements for the scheme are derived from representative use cases. A key requirement, also for legal validity of the signature, is the reconstruction of the signed form, when validating the signature. All the resources, referenced by the form, including client-side default stylesheets, have to be included within the signature. Finally, this paper presents an implementation of the scheme and a related use case. The author has designed and implemented the proposed solution. Also, he has written all of the text, while getting comments and proof-reading help from Prof. Petri Vuorimaa.
Author's overall contribution to the publication: 95 %.

Publication [P8]

This paper reviews two declarative user interface definition languages, XForms and XUL, from the point of view of desktop-style web applications. The paper presents a use case, which is an online journal editing system, and implements key parts from that use case using both languages. Requirements for UIDLs are gathered from related literature, and the languages are compared against the requirements. In order to fullfill the nested structures navigation requirement, an extension to XForms, the tree module, is proposed in this paper. The author has written half of the text and implemented the XForms version of the use case, while Pohja authored the XUL version. The author has also designed and implemented the XForms tree module. The use case was proposed by Ervamaa, and refined with wireframe models by Penttinen.
Author's overall contribution to the publication: 40 %.

Publication [P9]

This paper proposes a novel model, *XFormsMM*, for defining multimodal web applications. The model separates modality-independent parts from the modality-dependent parts. It includes a general *interaction manager*, which enables multimodal use of XForms UIs, and automatically synchronizes the modalities. The proposed model is compared to two other multimodal interaction authoring models, SALT and X+V, based on the Multimodal Interaction Requirements by W3C [71]. A prototype implementation of XFormsMM and the implemented use cases are described. The author has implemented and designed the multimodal interaction manager, and integrated the speech recognition and synthesizer libraries. He has also written 60 % of the text.
Author's overall contribution to the publication: 60 %.

BIBLIOGRAPHY

[1] Marc Abrams and James Helms. User interface markup language (UIML). UIML.org Language Specification, Version 3.0, February 2002.

[2] Marc Abrams, Constantinos Phanouriou, Alan L. Batongbacal, Stephen M. Williams, and Jonathan E. Shuster. Uiml: an appliance-independent xml user interface language. In *WWW '99: Proceeding of the eighth international conference on World Wide Web*, pages 1695–1708, New York, NY, USA, 1999. Elsevier North-Holland, Inc.

[3] Ali Arsanjani and David Chamberlain et al. (WSXL) web services experience language. IBM DeveloperWorks, 2002.

[4] Jonny Axelsson and et al. XHTML+Voice Profile 1.0. W3C Note, 2001.

[5] Pedro Azevedo, Roland Merrick, and Dave Roberts. OVID to AUIML - user-oriented interface modelling. TUPIS 2000, XML Cover Pages, 2000.

[6] Mark Bartel and et al. XML-Signature syntax and processing. W3C Recommendation, February 2002.

[7] John Barton, Tim Kindberg, Hui Dai, Nissanka B. Priyantha, and Fahd Al bin ali. Sensor-enhanced mobile web clients: an XForms approach. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 80–89, New York, NY, USA, 2003. ACM Press.

[8] Michel Beaudouin-Lafon. Instrumental interaction: an interaction model for designing post-wimp user interfaces. In *CHI '00: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 446–453. ACM Press, 2000.

[9] Tim Berners-Lee, Robert Cailliau, Ari Luotonen, Henrik Frystyk Nielsen, and Arthur Secret. The world-wide web. *Commun. ACM*, 37(8):76–82, 1994.

[10] Nina Bhatti, Anna Bouch, and Allan Kuchinsky. Integrating user-perceived quality into web server design. In *Proceedings of the 9th international World Wide Web conference on Computer networks : the international journal of computer and telecommunications netowrking*, pages 1–16, Amsterdam, The Netherlands, The Netherlands, 2000. North-Holland Publishing Co.

[11] Mark Birbeck. Building rich, encapsulated widgets using XBL, XForms and SVG. In *XTech Conference, available at : http://xtech06.usefulinc.com/schedule/paper/137, Accessed June 2006*, May 2006.

[12] Barclay Blair and John Boyer. XFDL: creating electronic commerce transaction records using xml. In *WWW '99: Proceeding of the eighth international conference on World Wide Web*, pages 1611–1622. Elsevier North-Holland, Inc., 1999.

[13] Barclay Blair and John Boyer. XFDL: Creating electronic transactions records using XML. In *Eighth Annual World Wide Web Conference (WWW8)*, 1999.

[14] Susanne Boll. *ZYX, Towards Flexible Multimedia Document Models for Reuse and Adaptation*. PhD thesis, University of Vienna, 2001.

[15] Susanne Boll. MM4U - a Framework for Creating Personalized Multimedia Content. In *Proceedings of the International Conference on Distributed Multimedia Systems*, Miami, Florida, September 24–26 2003. DMS.

[16] Matthias Book, Volker Gruhn, and Matthias Lehmann. Automatic dialog mask generation for device-independent web applications. In *ICWE '06: Proceedings of the 6th international conference on Web engineering*, pages 209–216, New York, NY, USA, 2006. ACM Press.

[17] John Boyer. Xforms & cause-and-effect programming. *Dr. Dobb's Journal*, March 2005.

[18] John Boyer, Tim Bray, and Maureen Gordon (eds.). Extensible forms description language (XFDL) 4.0. W3C Note, September 1998.

[19] John M. Boyer. Bulletproof business process automation: securing XML forms with document subset signatures. In *Proceedings of the 2003 ACM workshop on XML security*, pages 104–111. ACM Press, 2003.

[20] Alessandro Bozzon, Sara Comai, Piero Fraternali, and Giovanni Toffetti Carughi. Conceptual modeling and code generation for rich internet applications. In *ICWE '06: Proceedings of the 6th international conference on Web engineering*, pages 353–360, New York, NY, USA, 2006. ACM Press.

[21] Dick C. A. Bulterman and Lynda Hardman. Structured multimedia authoring. *ACM Trans. Multimedia Comput. Commun. Appl.*, 1(1):89–109, 2005.

[22] Gaëlle Calvary, Joëlle Coutaz, David Thevenin, Quentin Limbourg, Laurent Bouillon, and Jean Vanderdonckt. A unifying reference framework for multi-target user interfaces. *Interacting with Computers*, 15(3):289–308, 2003.

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

[24] P. Cesar. *A Software Architecture for High End Interactive Television Terminals*. Doctoral dissertation, Helsinki University of Technology, Finland, December 2005.

[25] Joyce Y. Chai, Pengyu Hong, and Michelle X. Zhou. A probabilistic approach to reference resolution in multimodal user interfaces. In *IUI '04: Proceedings of the 9th international conference on Intelligent user interface*, pages 70–77, New York, NY, USA, 2004. ACM Press.

[26] Philip R. Cohen. 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, 1992.

[27] Stijn Dekeyser, Jan Hidders, Richard Watson, and Ron Addie. Peer-to-peer form based web information systems. In *The Seventeenth Australasian Database Conference (ADC2006)*, 2006.

[28] Yogesh Deshpande, San Murugesan, Athula Ginige, Steve Hansen, Daniel Schwabe, Martin Gaedke, and Bebo White. Web engineering. *J.WEB ENGINEERING*, 1:003, 2002.

[29] Micah Dubinko. *XForms Essentials*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2003.

[30] Micah Dubinko and T.V. Raman (eds.). XForms 1.0. W3C Candidate Recommendation, 2003.

[31] Micah Dubinko and et al. (eds.). XForms 1.0. W3C Recommendation, 2003.

[32] Joshua Duhl. White paper: Rich internet applications. Technical report, IDC, 2003.

[33] ECMA-262m. ECMAScript language specification, 1998.

[34] David Hyatt (Editor). XML user interface language (XUL) 1.0. Mozilla.org, 2001.

[35] Carsten Eichholz, Anke Dittmar, and Peter Forbrig. Using task modelling concepts for achieving adaptive workflows. In *Engineering Human Computer Interaction and Interactive Systems: Joint Working Conferences EHCI-DSVIS*. Springer-Verlag GmbH, 2004.

[36] William K. English, Douglas C. Engelbart, and Melvyn L. Berman. Display-selection techniques for text manipulation. *Transactions on Human Factors in Electronics*, 1967.

[37] Arnaud Le Hors et al. Document object model (DOM) level 2 core specification - version 1.0. W3C Recommendation, November 2000.

[38] D. Visick et al. The use of simple speech recognizers in industrial applications. In *INTERACT'84*.

[39] Tom Pixley et al. Document object model (DOM) level 2 events specification. W3C Recommendation, November 2000.

[40] John Boyer et al. (eds.). XForms 1.1. W3C Working Draft, 2005.

[41] John Boyer et al. (eds.). XForms 1.0 (second edition). W3C Recommendation, 2006.

[42] Jonny Axelsson et al. (eds.). XHTML 2.0. W3C Working Draft, May 2005.

[43] Micah Dubinko et al. (eds.). XForms 1.0. W3C Recommendation, October 2003.

[44] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[45] Murielle Florins. *Graceful Degradation: a Method for Designing Multiplatform Graphical User Interfaces*. PhD thesis, UniversitÃ© catholique de Louvain, 2006.

[46] Murielle Florins and Jean Vanderdonckt. Graceful degradation of user interfaces as a design method for multiplatform systems. In *IUI '04: Proceedings of the 9th international conference on Intelligent user interface*, pages 140–147, New York, NY, USA, 2004. ACM Press.

[47] Martin R. Frank and Pedro A. Szekely. Adaptive forms: an interaction technique for entering structured data. *Knowledge-Based Systems*, 11(1):37–45, 1998.

[48] Piero Fraternali. Tools and approaches for developing data-intensive web applications: a survey. *ACM Comput. Surv.*, 31(3):227–263, 1999.

[49] Jesse Garrett. Ajax: A new approach to web applications. Technical report, Adaptive Path, 2005.

[50] Athula Ginige and San Murugesan. Web engineering, an introduction. *IEEE MultiMedia*, 8, 2001.

[51] Andreas Girgensohn, Beatrix Zimmermann, Alison Lee, Bart Burns, and Michael E. Atwood. Dynamic forms: an enhanced interaction abstraction based on forms. In *Human-Computer Interaction, INTERACT '95, IFIP TC13 Interantional Conference on Human-Computer Interaction*, pages 362–367, 1995.

[52] Robert L. Glass, Iris Vessey, and Venkataraman Ramesh. Research in software engineering: an analysis of the literature. *Information & Software Technology*, 44(8):491–506, 2002.

[53] Heng Guo. Implementation of secure web forms by using XML Signature and XForms. Master's thesis, Helsinki University of Technology, 2003.

[54] Rotan Hanrahan and Roland Merrick. Authoring techniques for device independence. W3C Working Group Note, February 2004.

[55] Michael Hoffman. Architecture of microsoft office infopath 2003. Microsoft Developer Network, October 2003.

[56] Simon Holland and Daniel Oppenheim. Direct combination. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 262–269. ACM Press, 1999.

[57] Philipp Hoschka et al. Synchronized multimedia integration language (SMIL) 2.0. W3C Recommendation, August 2001.

[58] Dave Hyatt. XBL - extensible binding language 1.0. Netscape, 2000.

[59] Hamid Jahankhani, John A. Lynch, and Jonathan Stephenson. The Current Legislation Covering E-learning Provisions for the Visually Impaired in the EU. In *EurAsia-ICT '02: Proceedings of the First EurAsian Conference on Information and Communication Technology*, pages 552–559, London, UK, 2002. Springer-Verlag.

[60] Kouichi Katsurada, Yusaku Nakamura, Hirobumi Yamada, and Tsuneo Nitta. XISL: a language for describing multimodal interaction scenarios. In *ICMI '03: Proceedings of the 5th international conference on Multimodal interfaces*, pages 281–284, New York, NY, USA, 2003. ACM Press.

[61] Alan Curtis Kay. *The reactive engine*. PhD thesis, 1969.

[62] Donald E. Knuth. *The Art of Computer Programming, Volume 1*. Addison-Wesley Professional, 1968.

[63] Sergei Kojarski and David H. Lorenz. Domain driven web development with webjinn. In *OOPSLA '03: Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 53–65, New York, NY, USA, 2003. ACM Press.

[64] Larry Koved and Ben Schneiderman. Embedded menus: selecting items in context. *Commun. ACM*, 29(4):312–318, 1986.

[65] Glenn E. Krasner and Stephen T. Pope. A cookbook for using the model-view controller user interface paradigm in smalltalk-80. *J. Object Oriented Program.*, 1(3):26–49, 1988.

[66] Tayeb Lemlouma and Nabil Layaida. Adapted content delivery for different contexts. In *International Symposium on Applications, and the Internet*, Orlando, Florida, January 27-31 2003.

[67] H. W. Lie, Tantek Celik, and Daniel Glazman. Media queries. W3C Candidate Recommendation, July 2002.

[68] Håkon W Lie. Formatting objects considered harmful. Web page, available at http://people.opera.com/howcome/1999/foch.html. Accessed June 2006, April 1999.

[69] Quentin Limbourg, Jean Vanderdonckt, Benjamin Michotte, Laurent Bouillon, and Víctor López-Jaquero. Usixml: A language supporting multi-path development of user interfaces. In *Engineering Human Computer Interaction and Interactive Systems, Joint Working Conferences EHCI-DSVIS 2004*, pages 200–220, 2004.

[70] Kris Luyten and Karin Coninx. An xml-based runtime user interface description language for mobile computing devices. In *Interactive Systems: Design, Specification, and Verification : 8th International Workshop, DSV-IS 2001*, Glasgow, Scotland, June 13-15 2001. Springer-Verlag GmbH.

[71] Stephane H. Maes and Vijay Saraswat (eds.). Multimodal interaction requirements. W3C NOTE, 2003.

[72] Marik Marshak and Hanoch Levy. Evaluating web user perceived latency using server side measurements. *Computer Communications*, 26(8):872–887, 2003.

[73] Gale Martin. The utility of speech input in user-computer interfaces. *International Journal of Man-Machine Studies*, 1989.

[74] Gavin F. McKenzie. XFA-FormCalc, version 1.0. W3C Note, 1999.

[75] Timur Mehrvarz, Daniel Appelquist, and Lasse Pajunen (eds.). WICD Core 1.0. W3C Working Draft, December 2005. http://www.w3.org/TR/WICD/.

[76] Pedro J. Molina, Santiago Meliá, and Oscar Pastor. User interface conceptual patterns. In *9th International Workshop on Interactive Systems. Design, Specification, and Verification, DSV-IS*, pages 159–172, 2002.

[77] Andreas Muller, Peter Forbrig, and Clemens H. Cap. Model-based user interface design using markup concepts. In *DSV-IS '01: Proceedings of the 8th International Workshop on Interactive Systems: Design, Specification, and Verification-Revised Papers*, pages 16–27, London, UK, 2001. Springer-Verlag.

[78] Brad Myers. User interface software tools. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 2(1):65–103, 1995.

[79] Brad Myers. Graphical user interface programming. In Allen B. Tucker, editor, *CRC Handbook of Computer Science and Engineering*, chapter 48. CRC Press, Inc., Boca Raton, FL (USA), second edition, 2004.

[80] Brad Myers, Scott E. Hudson, and Randy Pausch. Past, present, and future of user interface software tools. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 7(1):3–28, 2000.

[81] Theodor H. Nelson. Complex information processing: a file structure for the complex, the changing and the indeterminate. In *Proceedings of the 1965 20th national conference*, pages 84–100, New York, NY, USA, 1965. ACM Press.

[82] Jeffrey Nichols, Brad Myers, Thomas K. Harris, Roni Rosenfeld, Stefanie Shriver, Michael Higgins, and Joseph Hughes. 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, Washington, DC, USA, 2002. IEEE Computer Society.

[83] Dan R. Olsen. Interacting in chaos. *Interactions*, 6(5):42–54, 1999.

[84] Sharon Oviatt. Mutual disambiguation of recognition errors in a multimodel architecture. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 576–583, New York, NY, USA, 1999. ACM Press.

[85] Sharon Oviatt. Multimodal system processing in mobile environments. In *UIST '00: Proceedings of the 13th annual ACM symposium on User interface software and technology*, pages 21–30, New York, NY, USA, 2000. ACM Press.

[86] Sharon Oviatt. *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications*, chapter 14: Multimodal interfaces. Lawrence Erlbaum Assoc, 2003.

[87] Sharon Oviatt, Rachel Coulston, and Rebecca Lunsford. When do we interact multimodally?: cognitive load and multimodal communication patterns. In *ICMI '04: Proceedings of the 6th international conference on Multimodal interfaces*, pages 129–136, New York, NY, USA, 2004. ACM Press.

[88] Fabio Paterno and Carmen Santoro. One model, many interfaces. In *Proceedings of the 4th International Conference on Computer-Aided Design of User Interfaces CADUI2002*, 15-17 May 2002.

[89] Linda Dailey Paulson. Building rich web applications with Ajax. *Computer*, 38(10):14–17, 2005.

[90] Steven Pemberton et al. XHTML 1.0: The extensible hypertext markup language (2nd edition). W3C Recommendation, August 2002.

[91] Kari Pihkala. *Extensions to the SMIL Language*. Doctoral dissertation, Helsinki University of Technology, Finland, November 2003.

[92] Mikko Pohja and Petri Vuorimaa. CSS layout engine for compound documents. In *Proceedings of The third Latin American Web Congress (LA-WEB'05)*, November 2005.

[93] Juan Carlos Preciado, Marino Linaje Trigueros, F. Sanchez, and Sara Comai. Necessity of methodologies to model rich internet applications. In *Seventh IEEE International Symposium on Web Site Evolution (WSE*, pages 7–13, 2005.

[94] Angel Puerta. The mecano project: Comprehensive and integrated support for model-based user interface development. In *Proc. Of the 2nd Int. Workshop on Computer-Aided Design of User Interface CADUI96*, pages 19–37, Namur, June 5-7 1996. Presses Universitaires de Namur.

[95] Angel Puerta and Jacob Eisenstein. Ximl: a common representation for interaction data. In *IUI '02: Proceedings of the 7th international conference on Intelligent user interfaces*, pages 214–215, New York, NY, USA, 2002. ACM Press.

[96] Dave Ragget et al. HTML 4.0.1 specification. W3C Recommendation, December 1999.

[97] Dave Raggett and Max Froumentin. CSS Extensions for Multimodal Interaction. WWW page, 2004. Available online http://www.w3.org/2004/10/css-mmi/.

[98] T.V. Raman. *XML Powered Web Forms*. Addison-Wesley, 1st edition, 2003.

[99] JV. Ramesh, Robert L. Glass, and Iris Vessey. Research in computer science: an empirical study. *Journal of Systems and Software*, February 2004.

[100] Brent Rector. Introducing "longhorn" for developers. Microsoft Developer Network, October 2003.

[101] Lloyd Rutledge. Anticipating smil 2.0: The developing cooperative infrastructure for multimedia on the web. In *8th International World Wide Web Conference*.

[102] Patrick Schmitz. The smil 2.0 timing and synchronization model: Using time in documents. Technical report, MSR-TR-2001-01, Microsoft Corporation, 2001.

[103] Ben Schneiderman. Direct manipulation : a step beyond programming languages. *IEEE Computer*, 1983.

[104] Ben Schneiderman. *Designing the user interface: strategies for effective human-computer interaction, 2nd edition*. Addison-Wesley, 1992.

[105] Rainer Simon, Florian Wegscheider, and Konrad Tolar. Tool-supported single authoring for device independence and multimodality. In *MobileHCI '05: Proceedings of the 7th international conference on Human computer interaction with mobile devices & services*, pages 91–98, New York, NY, USA, 2005. ACM Press.

[106] David Canfield Smith, Charles Irby, Ralph Kimball, Bill Berplank, and Eric Harslem. Designing the star user interface. pages 237–259, 1990.

[107] Keith Smith. Simplifying Ajax-style web development. *Computer*, 39(5):98–101, 2006.

[108] Nathalie Souchon and Jean Vanderdonckt. Evaluating XML based service adaptation. In *Proceedings of the 10th International Workshop on Interactive Systems. Design, Specification, and Verification, DSV-IS*, Funchal, Madeira Island, Portugal, June 2003.

[109] Nathalie Souchon and Jean Vanderdonckt. A review of XML-compliant user interface description languages. In *Proceedings of the 10th International Conference on Design, Specification, and Verification of Interactive Systems*, pages 377–391, Madeira, Portugal, June 4-6 2003. Springer-Verlag.

[110] Nicole Stavness and Kevin Schneider. Supporting workflow in user interface description. In *Workshop on Developing User Interface Description Languages, AVI2004*, 2004.

[111] Warner ten Kate, Patrick Deunhouwer, and Ramon Clout. Timesheets - integrating timing in xml. In *Multimedia on the Web Workshop, 9th international World Wide Web Conference, WWW9*.

[112] Shari Trewin, Gottfried Zimmermann, and Gregg Vanderheiden. Abstract representations as a basis for usable user interfaces. *Interacting with Computers*, 16(3):477–506, 2004.

[113] Jacco van Ossenbruggen, Lynda Hardman, Joost Geurts, and Lloyd Rutledge. Towards a multimedia formatting vocabulary. In *Proceedings of the 12th International Conference on World Wide Web*, pages 384–393, Budapest, Hungary, May 20-24 2003. ACM.

[114] Ivan P. Velez and Bienvenido Velez. Lynx: An open email extension for workflow systems based on web services and its application to digital government. *aict-iciw*, 0:160, 2006.

[115] Kuansan Wang. Salt: A spoken language interface for web-based multimodal dialog systems. In *Proceedings of the 7th International Conference on Spoken Language Processing (ICSLP'02)*, 2002.

[116] Thomas Ziegert, Markus Lauff, and Lutz Heuser. Device independent web applications - the author once - display everywhere approach. In *Web Engineering - 4th International Conference, ICWE 2004, Munich, Germany, July 26-30, 2004, Proceedings*, pages 244–255, 2004.

[117] Gottfried Zimmermann and Gregg Vanderheiden. Technical requirements for a delivery context independent user interface model. W3C Workshop on Device Independent Authoring Techniques, Sep 2002.

[118] Gottfried Zimmermann, Gregg Vanderheiden, and Al Gilman. Prototype implementations for a universal remote console specification. In *CHI '02: CHI '02 extended abstracts on Human factors in computing systems*, pages 510–511, New York, NY, USA, 2002. ACM Press.

[119] Gottfried Zimmermann, Gregg Vanderheiden, Matthew Ma, Maribeth Gandy, Shari Trewin, Sharon Laskowski, and Mark Walker. Universal remote console standard: toward natural user interaction in ambient intelligence. In *CHI '04: CHI '04 extended abstracts on Human factors in computing systems*, pages 1608–1609, New York, NY, USA, 2004. ACM Press.