Aalto University
School of Science and Technology
Faculty of Electronics, Communications and Automation
Degree Programme of Communications Engineering

Markku Pekka Mikael Laine

XFormsDB—An XForms-Based Framework for Simplifying Web Application Development

Master's Thesis

Helsinki, Finland, January 20, 2010

Supervisor: Professor Petri Vuorimaa, D.Sc. (Tech.)

Instructor: Mikko Honkala, D.Sc. (Tech.)

Aalto University
School of Science and Technology
Faculty of Electronics, Communications and Automation
Degree Programme of Communications Engineering

ABSTRACT OF THE
MASTER'S THESIS

Author: Markku Pekka Mikael Laine

Title: XFormsDB—An XForms-Based Framework for Simplifying Web Application Development

Number of pages: xx + 161 **Date:** January 20, 2010

Language: English

Professorship: Interactive Digital Media and Contents Production

Code: T-111

Supervisor: Professor Petri Vuorimaa, D.Sc. (Tech.)

Instructor: Mikko Honkala, D.Sc. (Tech.)

The nature of the World Wide Web is constantly changing to meet the increasing demands of its users. While this trend towards more useful interactive services and applications has improved the utility and the user experience of the Web, it has also made the development of Web applications much more complex.

The main objective of this Thesis was to study how Web application development could be simplified by means of declarative programming. An extension that seamlessly integrates common server-side functionalities to the XForms markup language is proposed and its feasibility and capabilities are validated with a proof-of-concept implementation, called the XFormsDB framework, and two sample Web applications.

The results show that useful, highly interactive multi-user Web applications can be authored quickly and easily in a single document and under a single programming model using the XFormsDB framework.

Keywords: XFormsDB, XForms, XRX, XML, framework, WWW, Web, database, declarative

Aalto-yliopisto
Teknillinen korkeakoulu
Elektroniikan, tietoliikenteen ja automaation tiedekunta
Tietoliikennetekniikan koulutusohjelma

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä:	Markku Pekka Mikael Laine
Työn nimi:	XFormsDB – XForms-pohjainen ohjelmistokehys helpottamaan WWW-sovellusten kehittämistä
Sivumäärä:	xx + 161 **Päivämäärä:** 20. tammikuuta 2010
Julkaisukieli:	englanti
Professuuri:	Vuorovaikutteinen digitaalinen media ja sisällöntuotanto
Koodi:	T-111
Työn valvoja:	Professori Petri Vuorimaa, TkT
Työn ohjaaja:	TkT Mikko Honkala

WWW:n luonne muuttuu jatkuvasti vastatakseen paremmin käyttäjien kasvavia tarpeita. Vaikka tämä kehitys kohti hyödyllisempiä vuorovaikutteisia palveluita ja sovelluksia on parantanut WWW:n käyttö- ja käyttäjäkokemusta, niin se on myös samalla tehnyt WWW-sovellusten kehittämisestä paljon monimutkaisempaa.

Tämän työn päätavoitteena oli tutkia, miten WWW-sovellusten kehittämistä voitaisiin helpottaa deklaratiivisen ohjelmoinnin keinoin. Työssä esitetään laajennus, jonka avulla yleisimmät palvelinpään toiminnallisuudet voidaan saumattomasti liittää osaksi XForms-merkintäkieltä. Myös laajennuksen käyttökelpoisuus ja mahdollisuudet validoidaan prototyyppitoteutuksen, nimeltään XFormsDB-ohjelmistokehys, ja kahden WWW-esimerkkisovelluksen avulla.

Tulokset osoittavat, että XFormsDB-ohjelmistokehyksen avulla voidaan kirjoittaa hyödyllisiä, erittäin vuorovaikutteisia monen käyttäjän WWW-sovelluksia nopeasti ja helposti vain yhtä dokumenttia ja yhtä ohjelmointimallia käyttäen.

Avainsanat:	XFormsDB, XForms, XRX, XML, ohjelmistokehys, WWW, Web, tietokanta, deklaratiivinen

Acknowledgments

I would like to thank the following persons:

Dr. Mikko Honkala and Oskari Koskimies at NRC for suggesting such an interesting Master's Thesis topic and for sharing their invaluable expertise in XForms and related technologies. I would also like to express a special thank you to Dr. Mikko Honkala, my Thesis advisor, for his great guidance and support.

Professor Petri Vuorimaa at the Aalto University for pushing XFormsDB forward as well as for guiding and supervising my Thesis. In addition, I would like to thank all my co-workers at the Web Services research group—especially Pia Ojanen and Kalle Säilä for the laughs and the development of the XFormsDB framework, respectively.

Associate Professor Emilia Mendes for giving me an opportunity to join her research group at the University of Auckland, New Zealand for a couple of months. What a wonderful experience that was!

Friends and family for their endless help and support during my studies. I love you all, you are the ones who inspire me and make the life worth living.

Jenni Antikainen, to whom I am deeply grateful for always being there for me during all these years. I love you so much!!! ♥

This research was conducted as part of TIVIT's Flexible Services program and its Ecosystem Design and Evolution (EDEN) project, and was funded by the Finnish Funding Agency for Technology and Innovation (Tekes) and Nokia Research Center (NRC).

Helsinki, Finland, January 20, 2010

Markku Laine
markku.laine@gmail.com

Table of Contents

Abbreviations and Terms

ACID	Atomicity, Consistency, Isolation, and Durability (database transaction properties)
AJAX	Asynchronous JavaScript and XML
ANSI	American National Standards Institute
API	Application Programming Interface
AVT	Attribute Value Template
base64	An encoding method
cookie	A small piece of data stored in the user's user agent
CSS	Cascading Style Sheets
CSV	Comma Separated Values
DHTML	Dynamic HTML
DOM	Document Object Model
DSL	Digital Subscriber Line
DTD	Document Type Definition

ECMAScript	A scripting language
FLWOR	For, Let, Where, Order By, Return (XQuery)
FR	Framework Requirement
Framework	A re-usable design for a software system
hex	An encoding method
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IDE	Integrated Development Environment
Internet	A global, public computer network
Intranet	A private computer network
ISO	International Organization for Standardization
J2EE	Java 2 Platform, Enterprise Edition
JAR	Java Archive
Java	An object-oriented programming language
JavaScript	A scripting language
JDBC	Java Database Connectivity
JDK	Java Development Kit
JPEG	Joint Photographic Experts Group
JSP	JavaServer Pages
LR	Language Requirement

MB	Megabyte
MD	Message Digest
Middleware	A software that connects software components or applications
MIME	Multipurpose Internet Mail Extensions
MIP	Model Item Property
MVC	Model-View-Controller (architecture)
NXD	Native XML Database
OQL	Object Query Language
ORM	Object-Relational Mapping
PCDATA	Parsed Character Data
PDF	Portable Document Format
PIM	Personal Information Management
Plug-in	A software extension that enables added capabilities
PoC	Proof of Concept
RAM	Random Access Memory
RIA	Rich Internet Application
RDB	Relational Database
RDBMS	Relational Database Management System
RE	Requirements Engineering
REST	Representational State Transfer

RTF	Rich Text Format
SAX	Simple API for XML
SE	Software Engineering
Servlet	A Java program running on a Web server
SGML	Standard Generalized Markup Language
SHA	Secure Hash Algorithm
SQL	Structured Query Language
SQL/XML	SQL-based extensions for using XML in conjunction with SQL
SSL	Secure Sockets Layer
SVG	Scalable Vector Graphics
UI	User Interface
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
User agent	A software component (e.g., Web browser) running in the user's device
W3C	World Wide Web Consortium
WAR	Web Archive
Web	World Wide Web
WS	Web Services
WWW	World Wide Web

XFA	XML Forms Architecture
XFDL	Extensible Forms Description Language
XEDB	XML Enabled Database
XForms	An XML application representing the next generation of forms for the Web
XFormsDB	An XForms-based framework for simplifying Web application development
XHR	XMLHttpRequest (AJAX)
XHTML	Extensible HyperText Markup Language
XML	Extensible Markup Language
XMLNS	XML Namespace
XML-RPC	XML-Remote Procedure Call
XML Schema	Defines the structure, content, and semantics of an XML document
XPath	XML Path Language
XQJ	XQuery API for Java
XQL	XML Query Language
XQuery	XML Query
XQueryX	XML Syntax for XQuery
XRX	XForms/REST/XQuery (architecture)
XSL	Extensible Stylesheet Language
XSLT	Extensible Stylesheet Language Transformations

List of Tables

List of Figures

List of Listings

Chapter 1

Introduction

The role of the World Wide Web (WWW or Web) is constantly increasing in our everyday lives as more and more useful interactive services and applications are becoming available through the Web. These new generation applications, called Rich Internet Applications (RIA) [129], are not only highly interactive and provide multimedia contents, but also offer users the opportunity to communicate and collaborate with the responsiveness of desktop applications [128, 130].

This march towards RIAs has improved the utility and the user experience of the Web, but it has also significantly increased the complexity of developing Web applications. Cardone *et al.* list three main reasons for the complexity. First, dynamic Web pages are often generated on the fly, which makes application source code harder to understand and debugging more difficult. Second, dynamic Web pages often contain a mixture of markup languages, client-side scripting code, and server-side function calls, which makes application source code nearly unreadable and difficult to maintain. Third, the high number of tools, technologies, and techniques used in developing Web applications makes those applications complicated to design and fragile to deploy and run. [2] Other important reasons for the complexity of developing Web applications include: dependence on tool support, complex and tangled application structure that mixes application logic with User Interface (UI)

details as well as portability issues related to significant differences between browsers, browser versions, and experience gained with particular technologies and tools [131].

Even though these problems could be solved, there still exist some hurdles to overcome related to traditional Web application development. First, the difference in programming model in different tiers and the partition of application functionality across different tiers has several drawbacks especially from the developer's point of view. Second, even limited offline support is hard to accomplish. Third, multi-user support such as data synchronization and transactions need to be considered separately for each application. [3, 9]

One way of simplifying Web application development and reducing the skill set needed by developers is to unify the client-side and server-side programming under a single programming model [132]. This single programming model could be based on server-side concepts, in which both the user interface and application logic is programmed using a server-side programming language [4, 6] and the data source is integrated to the application logic using an object to data source schema mapping [7, 8]. Alternatively, it could be based on client-side concepts, such as using JavaScript [66] as a common language for the whole application, i.e., extending JavaScript with server-side functionalities [132]. When developing real-life Web applications, however, designing and implementing a user interface almost always requires human involvement and judgment, whereas most of the server-side functionalities can be handled by a generic server-side component. Therefore, using client-side concepts as the basis for this single programming model is preferable.

The next question is whether this single programming model should be based on declarative programming, which describes *what* should be done, or imperative programming, which describes *how* things should be done. Schmitz gives three primary reasons in favor of declarative programming over imperative programming. First, the bulk of Web content authors are not programmers so script or code is not an option for them. Second, creating interoperable authoring tools for non-declarative languages is nearly impossible. Third, without standard, declarative languages, the authoring and publishing environment becomes fragmented, which ultimately hurts the adoption and deployment in the marketplace. Furthermore, the use of declarative

programming provides a higher semantic level as well as is more secure, since the syntax is bounded. [5] Imperative programming languages, on the other hand, have more expressive power, but are generally harder to author and understand [19, 133]. In summary, in order to utilize the pre-existing skill set of Web content authors and to ease the adaptation of the new authoring paradigm, this single programming model should be based on declarative programming, which is much more accessible to Web content authors, who are used to HyperText Markup Language (HTML) [47] and Cascading Style Sheets (CSS) [53].

One conceptually promising approach in this direction that has emerged in the recent years is the XForms/REST/XQuery (XRX) architecture [30], which is based on the combination of three standards: XForms (on the client side) [35], Representational State Transfer ((REST) interfaces) [78], and XQuery (on the server side) [116]. XRX provides a simple and elegant zero translation Web application architecture that uses Extensible Markup Language (XML) [17] to store data in all tiers. eXist-db [69], for instance, offers a good example of how to develop end-to-end Web applications using the XRX architecture. Its approach is to use XQuery as a page template language, somewhat similar to JavaServer Pages (JSP) [135], and to provide common server-side functionalities through XQuery extension modules. Even though this approach allows Web content authors to stay within the XML world, it does not integrate seamlessly with XForms on the client side. Orbeon Forms [72], on the other hand, has extended XForms with UI controls and convenience features, but due to its internal Model-View-Controller (MVC) based architecture [34], all common server-side tasks are handled separately by its XML processors. Thus, what is needed is a way to naturally integrate server-side functionalities with the XForms markup language, which would also make it possible to develop entire Web applications using a single document.

This Thesis focuses on addressing the aforementioned issues by taking the idea proposed by Birbeck [134] to the next level, and presenting a powerful and extensible XRX framework, *XFormsDB*, to allow for the rapid development of useful, highly interactive multi-user Web applications. The Web applications that use the framework rely on a generic server-side component, which provides the integration services to heterogeneous data sources as well as common server-side functionalities through an extension to the XForms markup language.

1.1 Organization of the Thesis

The rest of the Thesis is organized as follows. Chapter 2 introduces common concepts of Web technologies with a focus on XML and XForms technologies. Chapter 3 gives a high-level overview of how to use XML with different types of databases. Together these two Chapters constitute the literature review, which gives background and motivation for the actual research conducted in this Thesis.

The second part of the Thesis starts by presenting the aims of this research, including research objectives and questions, research methods used, and scope of the research. Next, in Chapter 5, an extension to the XForms markup language is designed to meet the research goals. Then, in Chapter 6 and Chapter 7, a proof-of-concept implementation and two sample Web applications using the implementation were developed to validate the feasibility of the extension. Finally, in Chapter 8, the conclusions on the work done are drawn and suggestions for future work are presented.

Chapter 2

XML and Web Technologies

In this Chapter, an introduction to XML and Web technologies which are relevant to this Thesis is given. The basics of XML and its related technologies (DTD, XML Schema, DOM, the concept of XML parsers, and XSLT) are described. In addition, the evolution process of Web development techniques from (X)HTML to XForms is presented, paying particular attention to the XForms standard and how the technology can be extended and utilized with various browsers, including Internet Explorer[1], Firefox[2], and Safari[3].

2.1 XML

Extensible Markup Language (XML) [17], The World Wide Web Consortium (W3C)'s [126] Recommendation since February 1998, is a meta-language for describing data objects called XML documents. XML is a simplified subset of Standard Generalized Markup Language (SGML, ISO 8879:1986(E)) [60], which thus means that XML documents are also conforming SGML documents.

---

[1] Internet Explorer, Web browser, http://www.microsoft.com/uk/windows/products/winfamily/ie/default.mspx
[2] Firefox, Web browser, http://www.mozilla.com/firefox
[3] Safari, Web browser, http://www.apple.com/safari/

Similar to SGML, XML was designed to be a standard way of describing data for any purpose and to be used as a data exchange format on the Web and elsewhere. Thanks to the simple and interoperable nature of XML as well as its associated powerful standards and processing tools available, the technology has been adopted widely within the information technology industry—especially on the server side and in the form of Web Services (WS) [64].

XML documents are composed of storage units called *entities*, which contain either parsed or unparsed data. Parsed entity contains either character data or markup, whereas unparsed entity may contain text or other type of data, such as images or video. Markup encodes a description of the document's storage layout and logical structure.

Unlike HTML, another widely used markup language on the Web, XML does not use a set of predefined tags but allows the author to define their own tags. XML documents, however, cannot be constructed arbitrarily but they must be written according to the XML specification. An XML document that conforms to the XML specification is called *well-formed*. The structure and constraints on the contents of an XML document can optionally be defined, for instance, by Document Type Definition (DTD) [17] or its XML-based successor, XML Schema [61]. An XML document that complies with a particular DTD/XML Schema, in addition to being *well-formed*, is said to be *valid*. An example of a simple well-formed XML document is shown in Listing 1.

Listing 1: A well-formed XML document

```
<?xml version="1.0" encoding="UTF-8"?>
<message>
  <from>john.doe@example.com</from>
  <to>jane.doe@example.com</to>
  <subject>This is the subject</subject>
  <body type="text/plain">
    This is the body of an email message.
  </body>
</message>
```

In order to process and validate XML documents, an XML parser, also known as XML processor, is needed. An XML parser is a software component that reads and analyzes XML documents, after which it makes the information from those XML documents available to applications and programming languages, usually through a known Application Programming Interface (API), such as the W3C's Document Object Model (DOM) [62] or the SAX Project's Simple API for XML (SAX) [63]. The most important difference between DOM and SAX is that DOM maps an XML document into an internal, in-memory tree structure (cf. Figure 1), whereas SAX presents an XML document as a serialized event stream.

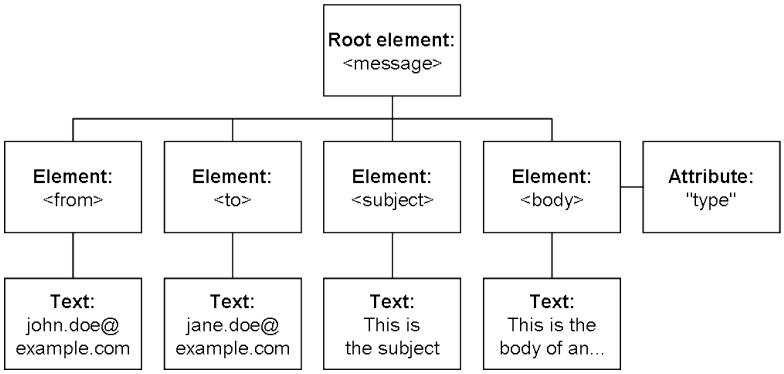

Figure 1: Listing 1 as an XML DOM node tree

2.1.1 XSLT

Extensible Stylesheet Language Transformations (XSLT) [56] is an XML-based stylesheet language which is designed to transform well-formed XML documents into some other forms, most commonly HTML, Extensible HyperText Markup Language (XHTML) [49], or another XML format [54]. The expressive power of XSLT is remarkable, since it is Turing complete, i.e., it is capable of performing any computational task [55]. XSLT has been a W3C Recommendation since November 1999 and it is the most important part of the Extensible Stylesheet Language (XSL) [57] family.

Figure 2 illustrates the XSLT transformation process, showing how an XML input, XSLT stylesheet, XSLT processor (e.g., Saxon [58]), and result tree are related to each other. In general, an XSLT processor takes two input files: an XML document

and an XSLT stylesheet. The XSLT processor then interprets and applies a set of declarative, template-based processing instructions found in the XSLT stylesheet to the XML document and finally outputs a new document called a result tree.

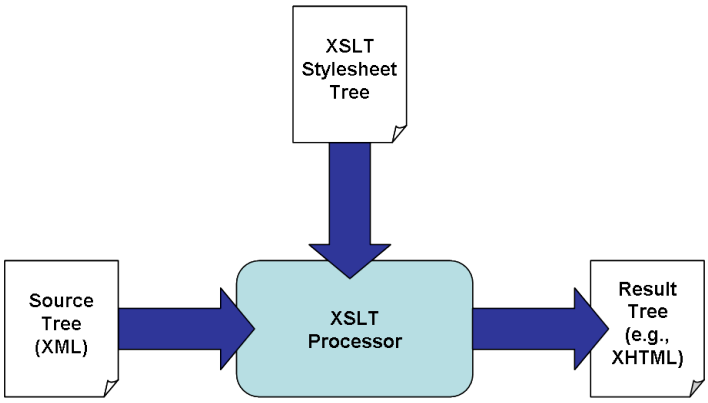

Figure 2: XSLT transformation process

XSLT transformation process can occur either on the client side or on the server side; although server-side transformations are dominant because not all browsers come with a built-in XSLT processor.

In the example below, an XSLT stylesheet (cf. Listing 2) is applied to the XML document, which was introduced in the previous Section (cf. Listing 1), and the result of the transformation is shown in Listing 3.

Listing 2: An XSLT stylesheet

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" version="1.0" />

  <xsl:template match="/">
    <emails>
      <xsl:for-each select="message/from | message/to">
        <xsl:sort select="." />
        <email><xsl:value-of select="." /></email>
      </xsl:for-each>
    </emails>
  </xsl:template>

</xsl:stylesheet>
```

Listing 3: The result of the transformation

```
<?xml version="1.0" encoding="UTF-8"?>
<emails>
  <email>jane.doe@example.com</email>
  <email>john.doe@example.com</email>
</emails>
```

2.2 (X)HTML

HTML

HyperText Markup Language (HTML) [47], which is based on SGML, is the primary markup language for creating Web pages on the Web. Its fundamental purpose is to provide a semantic description of the content and establish a document structure using hierarchical elements, such as headings, paragraphs, and lists. The first version of HTML was developed from the prototype written by Tim Berners-Lee at CERN in 1992 [45, 46]. The current versions of HTML in use today are defined in the W3C's HTML 4.01 Recommendation [47] and HTML 5 Working Draft [107].

XHTML

> *"Extensible HyperText Markup Language (XHTML) is a family of current and future document types and modules that reproduce, subset, and extend HTML, reformulated in XML. XHTML Family document types are all XML-based, and ultimately are designed to work in conjunction with XML-based user agents."* [48]

XHTML 1.0 [49] is a reformulation of HTML 4.01 according to the stricter syntax rules of XML. The elements are the same as in HTML, but there are some restrictions for document markup, such as all elements and attributes must be in lowercase. XHTML Basic [50], XHTML 1.1 [51], and XHTML 2.0 [52] again are module-based versions of XHTML, each one containing a certain set of modules targeted for special purposes, such as limited devices like mobile phones. The advantages of using XHTML over HTML are that XHTML is easier to author and maintain as well as it can be processed using a wide variety of XML tools, such as XSLT.

In both HTML and XHTML, the appearance of Web pages' content can be controlled through the use of Cascading Style Sheets (CSS) [53], which is a W3C standard for the visual presentation of Web pages. Unlike XSLT, CSS is not a Turing-complete language. However, this can be seen more as an advantage rather than a shortcoming, since CSS is easily analyzed and yet powerful enough for the purpose. [59]

2.2.1 Interaction Model

In the traditional (X)HMTL interaction model [41], which is illustrated in Figure 3, certain user actions in the interface (e.g., submitting an HTML form or clicking on a hyperlink) trigger an Hypertext Transfer Protocol (Secure) (HTTP(S)) [65] request to a server. The server processes the request (e.g., validates the submitted form data which is sent as name/value pairs) and then responds to the client by sending the result—normally in the form of a new (X)HTML document to be displayed within the browser.

The problem with this approach is that further user actions in the interface are suspended during the time the request is being processed on the server side. Furthermore, the approach wastes bandwidth because the entire Web page must be re-sent even if only part of it needs to be changed.

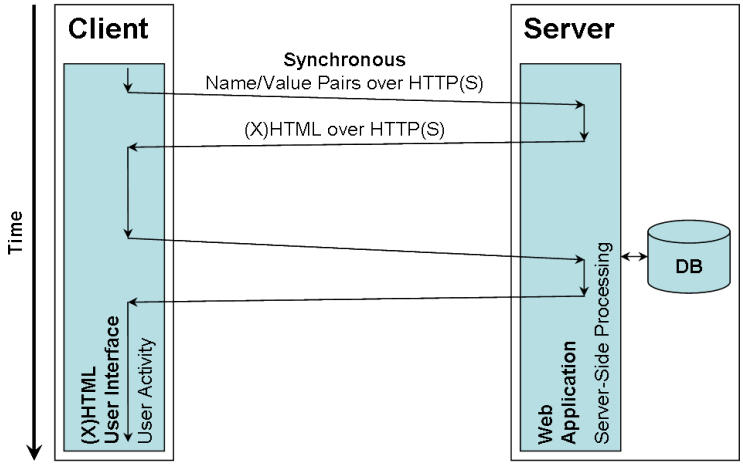

Figure 3: (X)HTML interaction model

2.3 AJAX

Asynchronous JavaScript and XML (AJAX), a term coined in 2005, is the Web development technique of the moment for creating richer and more interactive Web applications. AJAX is not a technology itself, but a combination of several existing technologies [41]:

- XHTML and CSS for standards-based presentation
- DOM for dynamic display and interaction
- XML and XSLT for data interchange and manipulation
- XMLHttpRequest (XHR) [42] for asynchronous data retrieval
- ECMAScript [66] (e.g., JavaScript) for binding everything together

The AJAX technique, and especially the XMLHttpRequest object, is supported by most common browsers, but with some differences in implementation. These differences, however, can be eliminated by using an abstraction library, such as Prototype JavaScript framework [43].

2.3.1 Interaction Model

In the AJAX interaction model [41], asynchronous submission, and other typical tasks such as serialization, deserialization, validation, and interaction, can all be executed on the client side by an AJAX engine, as depicted in Figure 4. The AJAX engine, which consists of an ECMAScript library, communicates with the user interface through DOM Level 2 Events [13] and ECMAScript handlers; and with the server by making asynchronous, behind the scenes, submissions using XML.

The advantage of using AJAX is that it eliminates the start-stop-start nature of interaction, which thus fixes the interaction problem existing in (X)HTML and its successor Dynamic HTML (DHTML) models. On the other hand, using AJAX brings on new problems, such as browser compatibility and accessibility issues as well as the need for browsers to support ECMAScript. However, some of these

problems (e.g., browser compatibility) will probably be fixed in the near future, whereas, for instance, AJAX accessibility issues will probably never be overcome. As a result, it is advised to develop two versions of a Web application; a version which uses AJAX and another version which does not use AJAX at all.

Google Suggest[1], for example, is a simple AJAX-based Web application, which uses AJAX to offer search term suggestions as you type.

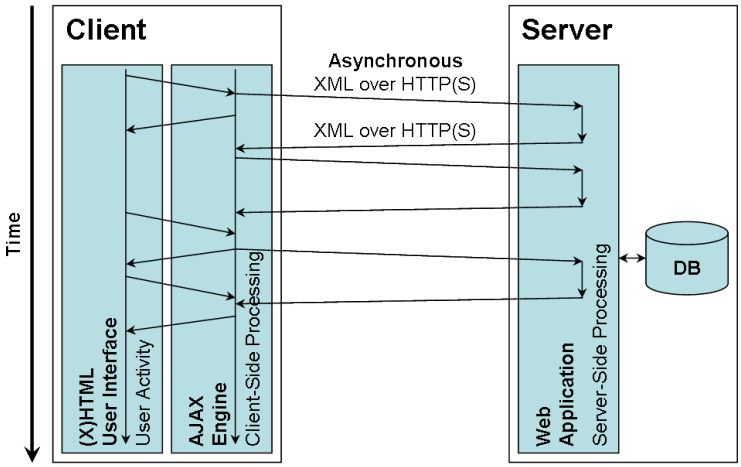

Figure 4: AJAX interaction model

[1] Google Suggest, Web application, http://www.google.com/webhp?complete=1&hl=en

2.4 XForms

XForms [35], which has been a W3C Recommendation since October 2003, is an XML-based forms technology and the successor to HTML forms. The design of XForms is based on former form technologies, such as Extensible Forms Description Language (XFDL) [37] and XML Forms Architecture (XFA) [38, 39], as well as conducting an in-depth analysis of HTML forms. The following list summarizes the primary benefits of using XForms:

- Separates data, logic, and presentation
- Integrates seamlessly with other XML technologies
- Platform, device, and modality independent
- Accommodates form component reuse
- Improves user experience: richer user interface and advanced forms logic
- Stores and transports data in XML documents
- Reduces or eliminates the need for scripting
- Will serve as the forms standard in XHTML 2.0
- Eases authoring of complex forms
- Fosters strong data type validation

The latest version of XForms, XForms 1.1 W3C Recommendation [40], refines the standard by adding several new features to the language, such as new and improved action handlers as well as more powerful action processing facilities for executing conditional actions and iterations, which all are essential for manipulating data arbitrarily. Thanks to these refinements, the information processing power of XForms 1.1 is now Turing complete.

Listing 4 shows a simple XHTML+XForms document which prints out the well-known phrase “Hello World!” to a browser window.

Listing 4: An XHTML+XForms document

```
<?xml version="1.0" encoding="UTF-8"?>
<html xml:lang="en" lang="en"
  xmlns="http://www.w3.org/1999/xhtml"
  xmlns:xforms="http://www.w3.org/2002/xforms">
  <head>
    <title>XForms in XHTML</title>
    <xforms:model>
      <xforms:instance id="data-instance">
        <data xmlns="">
          <phrase>Hello World!</phrase>
        </data>
      </xforms:instance>
    </xforms:model>
  </head>
  <body>
    <xforms:output ref="instance( 'data-instance' )/phrase" />
  </body>
</html>
```

2.4.1 Architecture

An XForms form uses the MVC architectural pattern that clearly separates the presentation from the data and logic of a form. The main pieces of an XForms form are illustrated in Figure 5.

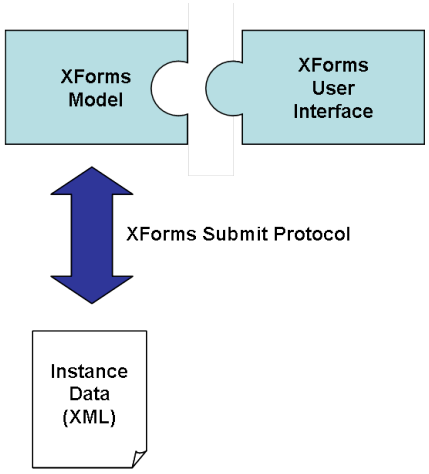

Figure 5: XForms architecture

Instance Data Defines the XML document template for data to be collected in a form. The initial content and structure of the XML document can be dynamically modified afterwards through user interactions.

XForms Model Declaratively defines the non-visual part of a form, i.e., the data and logic of a form. The data part is composed of one or more *Instance Data* definitions, whose structures and constraints can be defined using XML Schema. The logic part embodies data submission definitions and Model Item Properties (MIP) written in XML Path Language (XPath) [110]. The MIPs define dynamic calculations and constraints on *Instance Data* nodes (e.g., dependencies among various *Instance Data* nodes), which are not possible to define with XML Schema.

XForms User Interface *"The XForms User Interface provides a standard set of visual controls that are targeted toward replacing today's XHTML form controls. These form controls are directly usable inside XHTML and other XML documents, like SVG."* [33]

The XForms UI controls are bound to *Instance Data* nodes, which allow the separation of presentation and data.

XForms Submit Protocol Defines how XForms sends and receives *Instance Data* as well as how the data is being serialized. In general, the data is transferred to and from a server, but XForms also allows saving the data to local files, which can be reused later.

2.4.2 Interaction Model

The XForms interaction model clearly separates the user interface logic from the application logic, and thus reduces the need for round trips to the server as well as improves the user experience, since all of the user interface logic can be executed on the client side by an XForms processor.

Unlike in the AJAX interaction model, the user interface logic in the XForms interaction model is described declaratively, which raises the semantic level and allows adaptation to different devices and contexts [19].

The XForms language also allows asynchronous submission of forms (e.g., *Instance Data* in the XML format) to the server (cf. Figure 6), which enables the user interface to remain responsive, while the request is being processed on the server side.

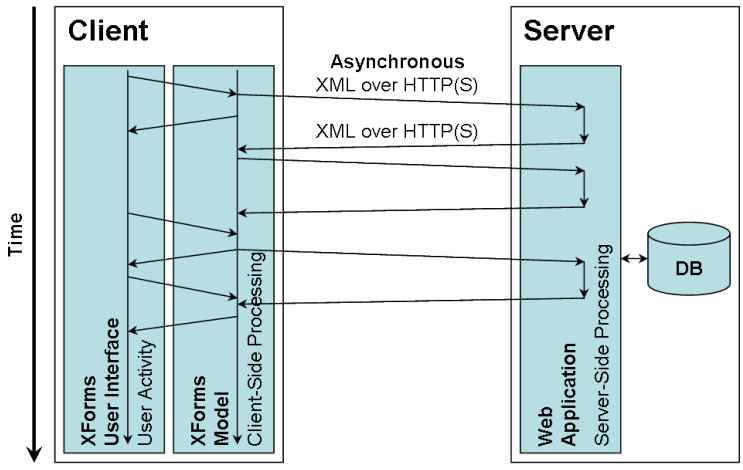

Figure 6: XForms interaction model

2.4.3 Implementations

Most common browsers still do not provide native XForms support, even though the W3C issued XForms as a W3C Recommendation as early as 2003.

Fortunately, supporting XForms natively in browsers is not the only option to use the XForms technology with existing browsers. Honkala has categorized the different XForms implementations into four groups [19]:

Native browser support Requested XHTML+XForms documents are sent to the user agent (e.g., browser) as static documents. The user agent is able to interpret and display the documents as such without requiring installation of additional software or plug-ins, or document transformations. In addition, the user agent allows XML to be used as the serialization format of instance data. For instance, X-Smiles [23, 24] supports XForms natively and it is one of the three XForms implementations referenced in the W3C XForms 1.0 Implementation Report [25].

Browser plug-in Since most common browsers have a rich plug-in interface, it opens the door for vendors to implement XForms as a browser plug-in, such as Mozilla XForms Project [26] and formsPlayer [27]. The browser plug-in needs to be installed on the user agent only once, after which the browser works as it would support XForms natively.

Another option for vendors is to utilize a popular plug-in already installed on most user agents. For instance, DENG [28] is a Flash-based XForms processor, which runs on any browser that has the Flash plug-in installed.

AJAX implementation Requested XHTML+XForms documents are sent to the user agent as static documents as in options 1 and 2. In addition, an XForms AJAX implementation, which translates XForms controls to and from standard (X)HTML controls on the client side, is sent along with the XHTML+XForms documents. This approach, which is demonstrated by FormFaces [29], does not require installation of additional software or plug-ins neither on the client side nor on the server side.

Server-side transformation Requested XHTML+XForms documents are transformed into plain (X)HTML+CSS or (X)HTML+CSS+ECMAScript on the server side before sending them to the user agent. Submissions are performed using either plain (X)HTML form submissions, in which the submitted form data is transformed to XML on the server side, or AJAX, respectively. This approach does not require installation of additional software or plug-ins on the user agent. On the server side, however, additional software (e.g., Orbeon Forms [72] or Chiba [31]) needs to be installed.

In the case of AJAXForms [32], which is a mixture of XForms server-side transformation and XForms AJAX implementation, the requested XHTML+XForms documents are first transformed on the server side, after which the XForms implementation works similar to XForms AJAX implementations.

The different XForms implementations are compared against a set of evaluation criteria, which is expanded from Honkala's Thesis (PhD) [19], as shown in Table 1. As can be seen, none of the XForms implementations is superior to others, and the choice between the different XForms implementations has to be made on the grounds of the project's main criteria. For instance, one of the main requirements of any common Web application is full cross-browser compatibility, which can be achieved only by using either an XForms AJAX implementation or an XForms server-side transformation. For a mobile phone manufacturer, however, browser independence is not an issue, but user interface latency and bandwidth consumption are the primary concern.

Table 1: Comparison of XForms implementations against a set of evaluation criteria. "+" means that a particular evaluation criterion does not have any negative effects on a particular XForms implementation, whereas "– " means the opposite.

Evaluation criterion / **XForms implementation**	**Client-side installation**	**Browser independent**	**ECMAScript independent**	**Server-side installation**	**Server-side technology independent**	**Offline support**	**User Interface latency**	**Bandwidth consumption**	**Amount of trusted components**
Native browser support	+	–	+	+	+	+	+	+	+
Browser plug-in	–	–	+	+	+	+	+	+	–
AJAX implementation	+	+	–	+	+	+	+ / –	–	+
Server-side transformation	+	+	+ / –	–	–	–	–	–	+

2.4.4 Extending XForms

Traditional HTML forms offer a limited amount of options for extensibility, whereas XForms has been explicitly designed with extensibility in mind. The following list contains distinct ways to extend XForms [18]:

Script One of the goals of XForms is to eliminate the need for a great deal of scripting. However, it would not be wise to replace an entire scripting language with declarative markup. Therefore, XForms provides a set of functions for accessing and updating *Instance Data* (a DOM document) through scripting[1].

New data types and libraries XForms incorporates W3C XML Schema data types and allows users to define their own application-specific data types, such as an email data type.

XPath extension functions XForms supports the use of XPath extension functions, which are defined using an XML namespace prefix. For example, eXforms [20] provides a set of useful functions to extend an XForms processor in a uniform way.

However, the downside of using this extension approach is that the XPath engine must implement the defined XPath extension functions, or otherwise an error is raised. Therefore, this extension approach is best suited to be used with XForms AJAX implementations or XForms server-side transformations.

New form controls XForms allows users to define their own application-specific form controls. However, the XForms processor must understand the special form controls in order to work properly that practically restricts the use of this extension approach to XForms AJAX implementations and XForms server-side transformations.

[1] XForms implementations, however, are not required to be based on the DOM.

XForms Actions Similar to new form controls, XForms allows users to define their own application-specific XForms Actions (e.g., a digital signature [21]) with the same XForms implementation restriction issues.

Custom events In addition to DOM-defined events and XForms-defined events, XForms provides a means for users to define their own application-specific events. The application-specific events can be sent off using the *xforms:dispatch* element and observed similar to other events.

New serialization formats An XForms processor can be extended to support new serialization formats by implementing the feature to be supported. The serialization format to be used is determined by both the *method* attribute and the URI Scheme used in the *action* attribute of the *xforms:submission* element.

Table 2 provides a summary of the distinct XForms extensions, which were described above, and how well they are suited to be used with different XForms implementations. As can be seen, both XForms AJAX implementation and XForms server-side transformation can be extended in all ways without requiring any updates to the user agent.

Table 2: Comparison of XForms implementations against a set of XForms extensions. "+" means that a particular XForms extension suits well to be used with a particular XForms implementation, whereas "–" means the opposite.

XForms extension / **XForms implementation**	**Script**	**New data types and libraries**	**XPath extension functions**	**New form controls**	**XForms Actions**	**Custom events**	**New serialization formats**
Native browser support	+	+	–	–	–	+	–
Browser plug-in	+	+	–	–	–	+	–
AJAX implementation	+	+	+	+	+	+	+
Server-side transformation	+	+	+	+	+	+	+

Chapter 3

XML and Databases

Chapter 2 covered the fundamentals of XML and Web technologies, paying particular attention to frontend technologies—especially to the XForms technology.

In this Chapter, the discussion on XML is continued from a backend perspective and a high-level overview of how to use XML with databases is given. First, the classification of XML documents based on their characteristics is discussed. After that, options for storing XML documents in different types of databases are examined. Finally, interfaces between XML documents and databases are presented.

3.1 Classification of XML Documents

XML documents are usually classified into three groups according to their content, structure, and supposed use: data-centric XML documents, document-centric XML documents, and hybrid XML documents. Characterizing and classifying XML documents to be used is essential as it determines what kind of database to use.

3.1.1 Data-Centric XML Documents

In data-centric XML documents, XML is used as an interchange format for data that is designed to be processed by a machine, rather than to be read by a human. As examples of data-centric XML documents may be mentioned flight schedules, stock quotes, and scientific data.

In his article, Bourret defines the characteristics of data-centric XML documents to be as follows [86]:

- Fairly regular structure
- Fine-grained data
- Little or no mixed content
- The order of sibling elements and PCDATA is generally not significant

As the example data-centric XML document in Listing 5 shows, changing the order of sibling elements does not change the meaning of the document.

Listing 5: A data-centric XML document

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
  <what>Research Group Meeting</what>
  <when>Friday August 28, 2009</when>
  <where>B122</where>
</note>
```

3.1.2 Document-Centric XML Documents

In document-centric XML documents, documents are usually designed for human consumption and written by hand in XML[1]. Examples are books, user guides, and almost any XHTML document.

The characteristics of document-centric XML documents are [86]:

- Less regular or irregular structure
- Larger grained data
- Lots of mixed content
- The order of sibling elements and PCDATA is almost always significant

As can be seen from Listing 6, the order of sibling elements is now very important for the meaning of the document.

Listing 6: A document-centric XML document

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
  Hi all, <what>Research Group Meeting</what> will be held on
  <when>Friday August 28, 2009</when> in room <where>B122</where>.
</note>
```

3.1.3 Hybrid XML Documents

Sometimes the distinction between data-centric XML documents and document-centric XML documents is not entirely clear. An XML document might contain characteristics of both XML document types (cf. Listing 7), in which case the document is said to be a hybrid XML document.

[1] Documents can also be written in other formats, such as Rich Text Format (RTF) or SGML, which are then converted to XML.

Listing 7: A hybrid XML document

```
<?xml version="1.0" encoding="UTF-8"?>
<message>
  <from>john.doe@example.com</from>
  <to>jane.doe@example.com</to>
  <subject>This is the subject</subject>
  <body type="text/plain">
    <note>
      Hi all, <what>Research Group Meeting</what> will be held on
      <when>Friday August 28, 2009</when> in room
      <where>B122</where>.
    </note>
  </body>
</message>
```

3.2 Options for XML Document Storage

The primary options for persistent data storage include databases and file systems. In this Section, the focus is on examining different types of databases only, as they provide multi-user support and assure Atomicity, Consistency, Isolation, and Durability (ACID) properties [36] as well as are the most popular data storage option for Web applications. Below, the following databases are examined in detail: Relational Databases (RDB), XML Enabled Databases (XEDB), and Native XML Databases (NXD).

3.2.1 Relational Databases

Relational Databases (RDB) are the most popular form of data storage in the world. As the name implies, they are based on a relational model, introduced by E.F. Codd at IBM Research Laboratory in 1970 [93].

In a Relational Database Management System (RDBMS), data is organized into two-dimensional tables (relations), consisting of rows (tuples) and columns (attributes), in which each cell (intersection of row and column) of the table contains only one simple value. The data stored in a RDBMS can be retrieved and manipulated by

using Structured Query Language (SQL) [94]. SQL is a comprehensive language for controlling and interacting with a RDBMS, and it is both an American National Standards Institute (ANSI) [95] and International Organization for Standardization (ISO) [96] standard.

The main problem with relational databases is that they are rigid because their only data structure is tables. In addition, they work only with limited, simple data types, such as integers, and thus have troubles handling complex and user-defined data types, such as XML. [97] Therefore, in order to transfer data between XML documents and relational databases an appropriate mapping procedure or a middleware must be used (cf. Section 3.3.1 and Section 3.3.5, respectively).

3.2.2 XML Enabled Databases

XML Enabled Databases (XEDB) are databases (e.g., relational databases) that have extended the basic database functionality to include XML data management capabilities. The extensions allow, among others, transferring data between XML documents and the internal model of a database without one having to worry about the difficulties of implementing a mapping layer or embedding a middleware oneself, as is often the case with relational databases when dealing with XML data.

XML:DB Initiative [75] has defined an XEDB as follows:

> *"A database that has an added XML mapping layer provided either by the database vendor or a third party. This mapping layer manages the storage and retrieval of XML data. Data that is mapped into the database is mapped into application specific formats and the original XML meta-data and structure may be lost. Data retrieved as XML is NOT guaranteed to have originated in XML form. Data manipulation may occur via either XML specific technologies (e.g., XPath, XSL-T, DOM or SAX) or other database technologies (e.g., SQL). The fundamental unit of storage in an XEDB is implementation dependent."* [87]

Similarly to relational databases, XEDBs are best suited for handling data-centric XML documents. In addition, XEDBs are also capable of handling document-centric

and hybrid XML documents, but only if the database provides a native XML storage and retrieval technology as well. [92]

All major database vendors, such as Oracle [91], IBM DB2 [90], and Microsoft SQL Server [89], have comprehensive XML support in their databases (cf. Section 3.3.2).

3.2.3 Native XML Databases

Native XML Databases (NXD) were created especially to overcome the shortcomings of relational databases when dealing with document-centric or hybrid XML documents.

The formal definition from XML:DB Initiative states that a NXD:

- *"Defines a (logical) model for an XML document -- as opposed to the data in that document -- and stores and retrieves documents according to that model. At a minimum, the model must include elements, attributes, PCDATA, and document order. Examples of such models are the XPath data model, the XML Infoset, and the models implied by the DOM and the events in SAX 1.0."* [87]
- *"Has an XML document as its fundamental unit of (logical) storage, just as a relational database has a row in a table as its fundamental unit of (logical) storage."* [87]
- *"Is not required to have any particular underlying physical storage model. For example, it can be built on a relational, hierarchical, or object-oriented database, or use a proprietary storage format such as indexed, compressed files."* [87]

Examples of NXDs are Software AG's commercial Tamino [88] and Wolfgang Meier's open source eXist-db [69].

The key advantage of using NXDs over other databases when dealing with document-centric or hybrid XML documents is that no data is lost, because no conversion is required between XML documents and the internal model of a

database. In addition, NXDs are better suited for querying and integrating data as well as handling schema changes and schemaless data.

Finally, NXDs generally support the same basic features as other databases, such as multi-user access, transactions, and locking—at least at the level of entire documents.

3.3 Interfaces between XML Documents and Databases

Most Web applications have persistent data of some sort and they use XML at some point during the process of transferring the data from a persistent data storage (e.g., database) to the Web application and vice versa. In this Section, five distinct approaches with code samples are presented for transferring data between XML documents and databases. In addition, an extensive middleware product called DataDirect XQuery is presented as a case example.

3.3.1 Mappings

The idea behind mappings is to map XML document schemas (e.g., DTDs or XML Schemas) to database schemas. Mappings are performed on element types, attributes, and text. In consequence of this, mappings almost always omit the physical and logical structure of an XML document, which thus makes them in general a poor choice for other than data-centric XML documents.

The main advantage of mappings is that they provide for a database-independent solution. They are also often used with XSLT in order to exactly match the structure expected.

Several methods have been proposed for mapping XML document schemas to database schemas. In his article, Bourret presents two commonly used mapping methods [86]:

- Table-based mapping
- Object-relational mapping (ORM)

Table-based mapping

Table-based mapping is a very simple mapping method, in which the XML document is modeled as a single table (cf. Listing 8) or a set of tables. It is widely implemented and used, for instance, by many middleware components for transferring data between XML documents and relational databases. This simple mapping method, however, has several disadvantages, such as it works only with a subset of XML documents, cannot handle mixed content at all, and does not preserve physical nor logical structure.

Listing 8: A table-based mapping for a single table

```
<?xml version="1.0" encoding="UTF-8"?>
<event>
  ...
  <row>
    <title>Research Group Meeting</title>
    <date>Friday August 28, 2009</date>
    <location>B122</location>
  </row>
  ...
</event>
```

<=>

Table: event

title	date	location
...	...	...
Research Group Meeting	Friday August 28, 2009	B122
...	...	...

Object-relational mapping (ORM)

Object-relational mapping (ORM) is a more sophisticated mapping method used by some middleware components and all XML enabled databases, in which the XML document is first modeled as a tree of objects and then the objects are mapped to a database (cf. Listing 9). ORM works for all XML documents, although it handles mixed content inefficiently and does not preserve physical nor logical structure. Because of this, ORM is a poor choice for document-centric XML documents.

Listing 9: An object-relational mapping

```
<?xml version="1.0" encoding="UTF-8"?>
<event>
  <title>Research Group Meeting</title>
  <date>Friday August 28, 2009</date>
  <location>B122</location>
</event>

                                  <=>

Object event {
  title    = "Research Group Meeting";
  date     = "Friday August 28, 2009";
  location = "B122";
}

                                  <=>

                           Table: event
```

title	date	location
...	...	...
Research Group Meeting	Friday August 28, 2009	B122
...	...	...

Unlike the name implies, ORM can also be used with non-relational databases, such as object-oriented and hierarchical databases. Therefore, a more appropriate name for the mapping method would be *object-based mapping* [86].

Alternative mapping methods

In addition to the aforementioned mapping methods, several other more advanced mapping methods have been proposed, such as the *Edge* and *Attribute* methods. In the *Edge* method, all elements and attributes with their values as well as parent-child relationships are stored as tuples in a single table called the *Edge* table. In the *Attribute* method, a similar table is created for *each* element or attribute name in the XML document. The *Edge* method performs poorly for heavy queries due to many joins with the large *Edge* table, whereas the *Attribute* method does not have this problem, because only relevant data is processed. [108]

3.3.2 Vendor-Specific XML Extensions

All major database vendors provide comprehensive XML support in one way or another. For instance, some databases implement a mapping layer or embed a middleware, whereas other databases provide XML extensions to SQL or support an XML query language. Furthermore, some non-native XML databases even provide native XML storage, and thus blur the line between native XML databases and XML enabled databases. Regardless of the used approaches, vendor-specific XML extensions are *not* an option when a database-independent solution is needed. [92]

In some cases, however, it might be justifiable to use vendor-specific XML extensions. For instance, when a company's software needs to be extended with XML data management capabilities and the software already runs atop an XML enabled database.

Finally, vendor-specific XML extensions have an effect on the following functionalities in a database, which must be taken into account when specifying software requirements: storage technology, indexing, flexibility, mapping, query language, updates, and performance.

3.3.3 SQL-Based Query Languages

SQL-based query languages use modified SELECT statements, whose results are transformed to XML. A number of proprietary SQL-based query languages exist and—as might be guessed—the solutions differ and are even based on different approaches. [86]

SQL/XML

In early 2000, an informal group of companies called SQLX [99] began to standardize XML extensions to SQL—the work which eventually led to the emergence of SQL/XML [98].

SQL/XML is an ANSI and ISO standard that provides XML extensions to SQL which, among others, include: (1) XML publishing functions, (2) the XML data type, and (3) mapping rules. By means of these extensions, it is possible to store XML documents in SQL database (e.g., relational database), to query those documents using XPath and XQuery, and to construct XML documents from existing SQL data (e.g., relational data). [99]

For a SQL programmer, SQL/XML is easy to learn because it is SQL-centric and it involves only a few additions to the familiar SQL language [101]. Another benefit of SQL/XML is that it can be used with traditional database APIs, such as Java Database Connectivity (JDBC) [100].

SQL/XML is supported by Oracle and IBM DB2, but not by Microsoft SQL Server. However, database-independent implementations of SQL/XML are also available, which can be used with any major relational database. [101]

Listing 10 shows a simple SQL/XML statement which is executed against the relational data presented in Listing 9. The result of the query is the XML document presented in Listing 5.

Listing 10: An SQL/XML statement

```
SELECT XMLELEMENT( NAME "note",
         XMLELEMENT( NAME "what", e.title ),
         XMLELEMENT( NAME "when", e.date  )
         XMLELEMENT( NAME "where", e.location ),
       )
FROM event e
WHERE e.title = "Research Group Meeting"
```

3.3.4 XML Query Languages

XPath

XML Path Language (XPath) [110], which is one of the W3C's core XML Recommendations, is an expression language for addressing parts of an XML document. XPath gets its name from its use of a path expression, which provides a means for navigating through the hierarchical structure of an XML document. In addition to path expressions, XPath encompasses a library of standard functions and operators.

The latest version, XPath 2.0, is widely implemented and used, either on its own or embedded in a host language, such as XSLT or XQuery.

Listing 11 shows a simple XPath expression which is executed against the virtual XML view of relational data presented in Listing 9. The result of the query is an *event* element that has a title with the value "Research Group Meeting".

Listing 11: An XPath expression

```
/event[ title = "Research Group Meeting" ]
```

XQuery

XQuery [116], a W3C Recommendation since January 2007, is a Turing complete XML query language applicable across all data sources that can be viewed as XML. It is derived from an XML query language called Quilt [111], which in turn was

influenced by several other languages, including XPath, XQL [112], XML-QL [113], SQL, and OQL [114].

XQuery and XPath are very closely related because XQuery 1.0 is an extension of XPath 2.0 and they both share the same data model [115] as well as the same set of functions and operators [118]. Because of this, any expression that is valid in both languages evaluates to the same value using both languages.

What makes XQuery much more powerful than XPath is that it overcomes the limitations of XPath (e.g., lack of grouping, sorting, and cross document joins) by providing a feature called a FLWOR expression, in which FLWOR stands for "*for*, *let*, *where*, *order by*, and *return*", the keywords used in the expression:

- The *for* clause iterates one or more variables over its binding sequence
- The optional *let* clause binds one or more variables to the result of its associated expression, without iteration
- The optional *where* clause serves as a filter for the tuples of variable bindings generated by the *for* and *let* clauses
- The optional *order by* clause contains one or more ordering specifications for reordering the tuples of variable bindings generated by the *for* and *let* clauses that satisfy the condition in the *where* clause
- The *return* clause forms the result of a FLWOR expression. It is evaluated once for each tuple of variable bindings generated by the *for* and *let* clauses that satisfy the condition in the *where* clause

Listing 12 shows a simple FLWOR expression which is executed against the virtual XML view of relational data presented in Listing 9. The result of the query is the XML document presented in Listing 5.

Listing 12: A FLWOR expression

```
for $event in /event[ title = "Research Group Meeting" ]
  return <note>
           <what>{ $event/title }</what>
           <when>{ $event/date }</when>
           <where>{ $event/location }</where>
         </note>
```

As can be seen from the above example, a FLWOR expression allows selecting and filtering XML data based on specific criteria as well as transforming the data into another XML vocabulary or structure. In addition, both built-in and user-defined functions can be used in FLWOR expressions, for example, for manipulating strings and dates as well as performing mathematical calculations.

One of the main design goals of XQuery was that it would use and share appropriate W3C Recommendations as much as possible, such as XML, Namespaces [124], and XML Schema. In addition, there are several peripheral standards that complement XQuery: XSLT 2.0 and XQuery 1.0 Serialization [117], XML Syntax for XQuery 1.0 (XQueryX) [119], XQuery Update Facility 1.0 [120], XQuery and XPath Full Text 1.0 [121], XQuery Scripting Extension 1.0 [122], and XQuery API for Java (XQJ) [123].

For a SQL programmer, XQuery requires more learning than SQL/XML because the language is new. For an XML programmer, however, the language is likely to be more natural because it is XML-centric and it fits cleanly into the XML world.

XQuery is widely implemented and supported by native XML databases as well as all major XML enabled database vendors.

3.3.5 Middleware

Middleware is a lightweight software component, which is commonly used for transferring data between data-centric XML documents and relational databases. Although middleware is usually used with relational databases, some middleware

products exist, which can be used for accessing data stored in other types of databases as well. [22]

Case example: DataDirect XQuery

DataDirect XQuery [125] is a commercial middleware component that provides both database and platform independent solution for querying and updating XML, relational data, legacy flat file data formats (e.g., Comma Separated Values (CSV)), or a combination of data sources.

The main benefit of DataDirect XQuery is that it uses a single query language—XQuery through XQJ—for accessing various data sources, including all major relational databases. An interesting detail about DataDirect XQuery is that if a relational database is queried, it decomposes the XQuery expression into highly optimized SQL statements in order to minimize the amount of data needed to be moved out of the database.

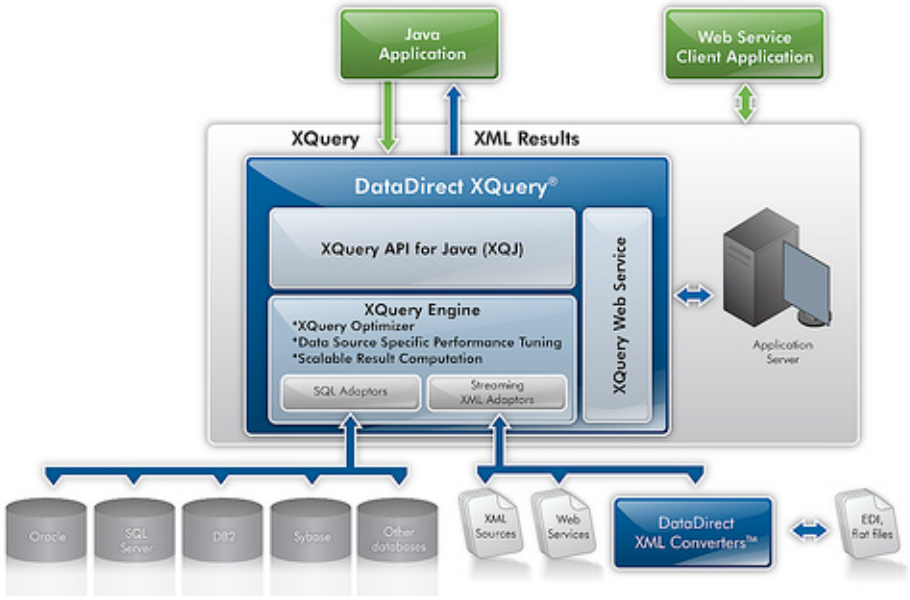

Figure 7: DataDirect XQuery architecture [125]

3.4 Summary

This Chapter covered the fundamentals of how to use XML with databases. The main conclusion is that the characteristics of XML documents define what kind of database should be used for storing the data. Relational databases and XML enabled databases, for instance, do not suit well for storing either document-centric XML documents or hybrid XML documents, as their fundamental unit of (logical) storage differs from XML, which in turn may result in lost of data. In addition, if data-centric XML documents need to be stored in relational databases, an appropriate mapping procedure or a middleware must be used. A detailed comparison of databases against XML document types and a set of interfaces are provided in Table 3 and Table 4.

Table 3: Comparison of databases against XML document types. "+" means that a particular XML document type suits well to be used with a particular database, whereas "–" means the opposite.

XML document type / **Database**	**Data-centric XML documents**	**Document-centric XML documents**	**Hybrid XML documents**
Relational Databases (RDB)	+	–	–
XML Enabled Databases (XEDB)	+	+ / –	+ / –
Native XML Databases (NXD)	+	+	+

Table 4: Comparison of databases against a set of interfaces. "+" means that a particular interface suits well to be used with a particular database, whereas "–" means the opposite.

Interface / Database	Mappings	Vendor-specific XML extensions	SQL-based query languages	XML query languages	Middleware
Relational Databases (RDB)	+	–	–	–	+
XML Enabled Databases (XEDB)	+	+	+	+ / –	+
Native XML Databases (NXD)	+	–	–	+	+

Chapter 4

Research Aims

4.1 Research Objectives and Scope

The main research objectives of this Thesis have been divided into two distinct parts. In the first part, the objective is to design an extension to the XForms markup language, which enables software developers and even non-programmers to build useful, highly interactive multi-user Web applications quickly and easily using purely declarative languages. The objective of the second part is to validate the feasibility of the designed extension by implementing a proof-of-concept prototype called the XFormsDB framework, which supports all the features of the designed extension. In addition, the reliability and usefulness of the framework has to be tested by developing useful real-life Web applications.

However, it must be emphasized that the goal of this Thesis is limited to implementing a working prototype of the framework, not a framework ready for production. Therefore, the efficiency and performance issues of the implementation are out of the scope, although they can be optimized for the final product. The framework will be designed taking non-programmers into account. However, the efficiency and feasibility will not be analyzed by extensive usability tests due to limited resources of the project.

4.2 Research Questions

The main research questions of this Thesis are the following:

Q1: Is it possible to extend the XForms markup language in such a way that users can build useful, highly interactive multi-user Web applications quickly and easily using purely declarative languages?

XForms provides a declarative markup language for authoring highly interactive, form-based Web applications. The architecture of XForms promotes building modular software by separating application logic, data, and presentation into distinct modules, and thus making XForms highly suitable for Web application development under a single model.

However, XForms was not intended as an end-to-end solution, but rather simply a client-side technology to be used with a separate server-side component. In general, a server-side component is mainly used for accessing a data source. By examining typical Web applications more closely, it can be noted that almost all of the functionality the server-side component provides for the client-side component is related to the data source access in one way or another: authentication and access control define what parts of the data source the user is allowed to access, session management provides the context and authorization for data source queries, and multi-user support is implemented by data source transactions. [9]

What if all the aforementioned functionalities could be defined in XForms? Does it mean that a server-side component can completely be left out and entire Web applications could be authored using a single document? These research questions are further discussed in Chapter 5 and Chapter 6 along with a proposed solution.

Q2: How the extension can be kept simple enough, so that even non-programmers are capable of utilizing it?

Even if a solution for extending the XForms markup language with server-side functionalities exists, it still might be too complex to be used by non-programmers. On the other hand, if the extension is too simple, it might not be expressive enough for advanced users, such as software developers. Could the extension be flexible enough to be used by both software developers and non-programmers?

From the author's point of view, there are several points which define the complexity of the extension, including suitability into the XForms programming model, intuitiveness in use, and possibility to use ready-made components authored by other users. Can these issues be solved, and thus resulting an extension which is both simple and flexible to use by all types of authors? These research questions are closely related to *Q1* and are also discussed in Chapter 5 and Chapter 6.

Q3: By what means should the feasibility of the extension be validated?

For making good research, it is not enough just to present results, but the results also need to be backed up with clean and convincing evidences. On what should these evidences be based on? How can they be determined?

Conceivably, the most obvious way to validate the designed extension is to implement it. On the other hand, the implementation only verifies that the designed extension is technically implementable; it does not validate the feasibility of the extension when authoring Web applications. Thus, a more extensive set of validation techniques is needed. Chapter 6 and Chapter 7 focus on the validation problem, whereas Chapter 8 evaluates the work as a whole.

4.3 Research Strategy

Based on the research objectives and scope as well as research questions presented above, an appropriate research strategy for solving the research problem needs to be found.

Shaw discusses the characteristics of good research in the area of Software Engineering (SE) and presents distinct research strategies based on the examination of several research papers. The presented research strategies have been constructed by selecting suitable combinations of the different types of research questions, results, and validation techniques. [10]

The research strategy of this Thesis is formed by following the guidance provided by the aforementioned paper and it can be recognized as a combination of the following types: *method or means of development* or *feasibility* (research questions), *notation or tool* (research results), and *example* (validation techniques). The recognized research strategy was also found among the research papers examined by Shaw, and thereby verifies the reliability of the selected research strategy [10].

Chapter 5

Design of the XFormsDB Markup Language

This Chapter describes the XFormsDB markup language, which was designed to meet the research objectives and to answer the research questions *Q1* and *Q2* presented in Chapter 4. The Chapter starts by collecting the list of requirements for the XFormsDB markup language. Then, all features as well as elements and attributes of the XFormsDB markup language are described in detail. Finally, evaluation of the designed language with respect to the requirements is carried out.

The complete syntax definitions and usage examples of the XFormsDB markup language are given in Appendix A.

5.1 Requirements

In Requirements Engineering (RE), the requirements for a system describe the services provided by the system and its operational constraints. According to Sommerville, requirements are categorized into two different levels of detail as follows: user requirements and system requirements. The level of detail to be used depends on the type of reader to whom the specification is targeted at. For instance, system end-users and contractor managers need a high-level statement of the

requirements, whereas software developers need a more detailed description of the requirements. [1]

Software system requirements are often classified into three categories: functional requirements, non-functional requirements, and domain requirements [1]. In general, requirements for markup languages are functional requirements and they are derived from usage scenarios [11, 12]. Therefore, usage scenarios were also used as the basis for deriving a detailed list of the main functional as well as non-functional requirements for the XFormsDB markup language (hereinafter referred to as the Language Requirements, LR) presented in Table 5.

Table 5: Requirements for the XFormsDB markup language

ID	Requirement
LR1	**The syntax and processing model of the XFormsDB markup language must be similar to XForms.** XForms uses declarative markup for describing operations in form-based Web applications. Thus, in order to naturally integrate the XFormsDB extension to XForms, the syntax and processing model of the XFormsDB markup language must resemble XForms.
LR2	**The architecture of the XFormsDB markup language must be easily extensible.** Web applications and user needs are constantly evolving, whereupon requirements for enhancements and new features often emerge over time. Therefore, the architecture of the XFormsDB markup language must provide a means for adding new features (XFormsDB-related requests) to the language while retaining the same processing model.

LR3	**XFormsDB-related requests must be able to be executed multiple times and at any point in the lifetime of a form.** XForms interaction model allows asynchronous communication with the server, meaning that submissions can take place multiple times and at any point in the lifetime of a form. Submitting XFormsDB-related requests must fulfill the same conditions.
LR4	**The XFormsDB markup language must provide a means for notifying XFormsDB-related request errors including detailed error messages.** Inevitably, problems can happen during XFormsDB-related request processing. In such situations, a notification with a detailed, understandable error message must be provided, for example “Failed to connect to data source”.
LR5	**The XFormsDB markup language must provide a means for facilitating modularity in XHTML+XFormsDB documents.** The level of XHTML+XFormsDB documents can vary from simple to extremely complex. Authoring of complex documents is also a time-consuming and error-prone operation. It must be possible to reuse ready-made components (e.g., user interface parts and queries) to decrease authoring errors and to speed up form authoring.

LR6	**The XFormsDB markup language must provide a means for maintaining state in XFormsDB Web applications.** In general, dynamic user interfaces need to keep track of the state information of a Web application in order to work properly. In the case of XForms, the state information of a Web application is stored into instance(s) and the user interface is dynamically presented according to that data. This approach, however, can be utilized only when the Web application consists of a single XHTML+XForms document, because XForms does not provide a means for passing information (e.g., instance data) between XHTML+XForms documents. For this reason, a mechanism for passing state information between XHTML+XFormsDB documents must be provided.
LR7	**The XFormsDB markup language must provide a uniform API for connecting to different types of data sources.** Each data source has a unique way of establishing a connection to the system. Therefore, in order to uniform the way of establishing a connection to a data source, an abstraction layer on the language level must be provided.

LR8	**The XFormsDB markup language must provide a mechanism for authentication and access control.** Most Web applications require an authentication mechanism to verify the identity of a user and to restrict user access to certain parts of the Web application, such as administration interface. In general, user authentication in Web applications is accomplished by one of the three mechanisms: HTTP authentication, Secure Sockets Layer (SSL) certificates, or form-based credentials [15]. However, none of these mechanisms integrate perfectly with the XForms programming model or they have significant shortcomings, such as poor user experience, complex to implement, lack of logout, no encryption/security, or require server-side programming. XForms 1.1 addresses some of the shortcomings described above by indirectly supporting Basic and Digest authentication with HTTP(S) but does not really provide a comprehensive and easy-to-use solution to overcome all of the problems related to authentication [16]. For this reason, the XFormsDB markup language must provide a simple way for form authors to authenticate users and to handle common tasks related to access control.
LR9	**The XFormsDB markup language must support a standardized query language applicable across different types of XML data sources.** XForms is designed to gather and process data in the XML format and it is intended to be used with other XML technologies. Because of this, the query language must be intended to be used with other XML technologies as well. Moreover, it should be standardized and widely supported to ensure its usefulness across a number of users as well as different products, such as middleware and database systems. In addition, the XFormsDB markup language must also provide a means for binding parameters to external variables used in query expressions.

LR10	**The XFormsDB markup language must provide a means for supporting multi-user concurrency.** In multi-user Web applications, users' interactions can easily conflict. For instance, several users may try to perform updates simultaneously. Because query languages do not provide a built-in, easy-to-use solution to this problem, a different approach for supporting multi-user concurrency must be provided.
LR11	**Security issues must be considered carefully in the design of the XFormsDB markup language.** In traditional Web applications, sensitive information (e.g., query expressions and data source configurations) is processed on the server-side to ensure that the sensitive information is neither exposed to nor cannot be altered by malicious clients. It is highly important that XHTML+XForms documents, which are sent to the client, do not either expose or allow unauthorized altering of sensitive information.

5.2 Namespace for XFormsDB

The namespace Uniform Resource Identifier (URI) for XFormsDB is *http://www.tml.tkk.fi/2007/xformsdb* and the namespace prefix associated with it is *xformsdb*, which is used throughout this Thesis. This, however, is only a convention meaning that any namespace prefix for XFormsDB may be used in practice.

5.3 The *xformsdb:instance* Element

The *xformsdb:instance* element is a new element that acts as a wrapper for all XFormsDB-related requests to be submitted. The benefit of using a wrapper around requests is that it enables adding new requests to the XFormsDB markup language without requiring any changes to the request submission process.

The functionality of the *xformsdb:instance* element is identical to the *xforms:instance* element with the exception that only certain parts of the instance data, depending on the type of the request, are allowed to be altered.

The different types of requests are described in detail in the following Subsections.

5.3.1 The *state* Request

The *state* request provides a means for passing a Web application's state information from one XHTML+XFormsDB document to another. An instance containing a Web application's state information can be stored in an XFormsDB implementation for the duration of the session and it can be later retrieved either by the same or different XHTML+XFormsDB documents.

The *xformsdb:state* Element

Required child element of the *xformsdb:instance* element specifying only the name of the request.

5.3.2 The *login* Request

The *login* request enables a user to authenticate to a Web application, after which the user can access to restricted parts of the Web application. The user authentication is performed by submitting a username and password combination to an XFormsDB implementation, which checks the privileges of the user against a realm (cf. Section 5.8) and stores the user's credentials to its credentials store for future reference upon a successful login.

The *xformsdb:login* Element

Required child element of the *xformsdb:instance* element that wraps two *xformsdb:var* elements; one for the *username* variable and the other for the *password* variable.

Table 6: Attributes of the *xformsdb:login* element associated with the *login* request

Attribute	Description
datasrc	Optional attribute specifying the ID of a data source configuration to be used by an XFormsDB implementation for connecting to the data source (realm). In the absence of this attribute, the default data source configuration of an XFormsDB implementation is used.
doc	Optional attribute specifying the name of an XML document for limiting authentication queries of a data source connection to a single *xformsdb_users.xml* document. Useful when a data source connection points to a collection of documents. Default value is an empty string.

The *xformsdb:var* Element

Two required child elements of the *xformsdb:login* element; one having *username* and the other one having *password* as the value of the *name* attribute of this element. The values of these elements are bound to the corresponding *xforms:input* and *xforms:secret* form controls of a login form.

Table 7: Attributes of the *xformsdb:var* element associated with the *login* request

Attribute	Description
name	Required attribute specifying the name (*username* or *password*) of a variable to which the form control (correspondingly *xforms:input* and *xforms:secret*) of a login form is bound to. Default value is an empty string.

5.3.3 The *logout* Request

The *logout* request enables a user to exit a Web application, after which the user cannot access to restricted parts of the Web application. As a result of a successful logout, an XFormsDB implementation removes the user's credentials from its credentials store as well as all other user-related information from the session.

The *xformsdb:logout* Element

Required child element of the *xformsdb:instance* element specifying only the name of the request.

5.3.4 The *user* Request

The *user* request provides a means for extracting information about the currently logged-in user, such as *username* and *roles* the user belongs to.

The *xformsdb:user* Element

Required child element of the *xformsdb:instance* element specifying only the name of the request.

5.3.5 The *query* Request

The *query* request defines a query to be executed against a data source upon a corresponding submission is dispatched. The query expression can be written either using XQuery or XPath, which are both W3C-defined standards for querying collections of XML data. In addition, the query can be parameterized, too.

XQuery expressions are used for retrieving data, creating new structures (e.g., joins), and updating data *without* data synchronization. XPath expressions, on the other hand, provide much simpler but less powerful means for retrieving and updating data (an XML fragment) *with* data synchronization.

The *xformsdb:query* Element

Required child element of the *xformsdb:instance* element that wraps necessary elements for specifying a query.

Table 8: Attributes of the *xformsdb:query* element associated with the *query* request

Attribute	Description
datasrc	Optional attribute specifying the ID of a data source configuration to be used by an XFormsDB implementation for connecting to the data source. In the absence of this attribute, the default data source configuration of an XFormsDB implementation is used.
doc	Optional attribute specifying the name of an XML document for limiting queries of a data source connection to a single XML document. Useful when a data source connection points to a collection of documents. Default value is an empty string.

The *xformsdb:expression* Element

Required child element of the *xformsdb:query* element containing a query expression either in XQuery (*select* and *all* expression types) or XPath (a combination of *select* and *update* expression types). The query expression can be written either inline in this element or to an external file referenced by the *resource* attribute.

Table 9: Attributes of the *xformsdb:expression* element associated with the *query* request

Attribute	Description
resource	Optional attribute indicating the URI of an XQuery or XPath expression. Behavior of relative URIs in links is determined by the host language, i.e., the form. Default value is an empty string.

The *xformsdb:xmlns* Element

Optional child element of the *xformsdb:query* element declaring an XML Namespace that is used in a query expression (XPath).

Table 10: Attributes of the *xformsdb:xmlns* element associated with the *query* request

Attribute	Description
prefix	Required attribute specifying the prefix of an XML Namespace. Default value is an empty string.
uri	Required attribute specifying the URI of an XML Namespace. Default value is an empty string.

The *xformsdb:var* Element

Optional child element(s) of the *xformsdb:query* element, whose value is linked to an external variable declared in an XQuery expression or used in an XPath expression. The variable in the XQuery or XPath expression must have the same name as the one specified in the *name* attribute of this element.

Table 11: Attributes of the *xformsdb:var* element associated with the *query* request

Attribute	Description
name	Required attribute specifying the name of an external variable declared in an XQuery expression or used in an XPath expression. Default value is an empty string.

The *xformsdb:secvar* Element

Optional child element(s) of the *xformsdb:query* element, which securely links the username or the space-separated list of the role names of the currently logged-in user to the external variable (*username* or *roles*, respectively) declared in an XQuery expression or used in an XPath expression. The variable in the XQuery or XPath expression must have the same name (*username* or *roles*) as the one specified in the *name* attribute of this element. The value of this element is not allowed to, and cannot, be altered because it is securely set on the server side by an XFormsDB implementation.

Table 12: Attributes of the *xformsdb:secvar* element associated with the *query* request

Attribute	Description
name	Required attribute specifying the name (*username* or *roles*) of a secured, external variable (*username* or *roles*) declared in an XQuery expression or used in an XPath expression. Default value is an empty string.

Synchronized updates

XFormsDB provides a simple and elegant way for updating and synchronizing data to be stored in a data source. The updating process *with* data synchronization includes two steps. In the first step, an XML fragment is retrieved from a data source using an XPath expression pointing to the root element of the XML fragment to be updated. The retrieved XML fragment can then be altered including deleting and inserting nodes, after which in the second step, the altered XML fragment is submitted back to be stored in the data source using the same XPath expression as before. Finally, an XFormsDB implementation returns the stored XML fragment, which may contain changes made by other clients, upon a successful submission.

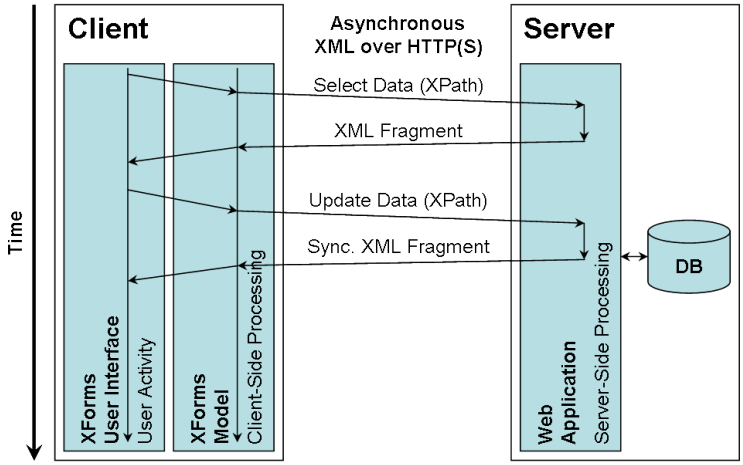

Figure 8: XFormsDB updating process *with* data synchronization

5.3.6 The *file* Request

The *file* request enables the users of a Web application to manage (select, update, insert/upload, delete, and download) files stored either within the Web application or to another location on the server. The Multipurpose Internet Mail Extensions (MIME) type of a file to be managed can be anything ranging from a Joint Photographic Experts Group (JPEG) image to a Portable Document Format (PDF) application.

The *xformsdb:file* Element

Required child element of the *xformsdb:instance* element that wraps necessary elements for performing a desired operation on one or more files.

The *xformsdb:var* Element

Optional child element of the *xformsdb:file* element, whose value (username, the space-separated list of file IDs, or the space-separated list of role names) is linked to an appropriate variable used by the *file* request to filter the list of files to be selected.

Table 13: Attributes of the *xformsdb:var* element associated with the *file* request

Attribute	Description
name	Required attribute specifying the name (*username*, *ids* or *roles*) of a variable used by the *file* request to filter the list of files to be selected. Default value is an empty string.

The *xformsdb:secvar* Element

Optional child element of the *xformsdb:file* element, whose value (username of the currently logged-in user or the roles of the currently logged-in user) is securely linked to an appropriate variable used by the *file* request to filter the list of files to be selected. The value of this element is not allowed to, and cannot, be altered because it is securely set on the server side by an XFormsDB implementation.

Table 14: Attributes of the *xformsdb:secvar* element associated with the *file* request

Attribute	Description
name	Required attribute specifying the name (*username* or *roles*) of a secured variable used by the *file* request to filter the list of files to be selected. Default value is an empty string.

Operations on files

XFormsDB provides four operations for managing files. The *select* operation enables retrieving the metadata about files including the URI from which the file can be downloaded. The *insert*, *delete*, and *update* operations, however, differ from the *select* operation and the idea behind them is similar to each other. In each operation, required information about the files is submitted to an XFormsDB implementation within an appropriate wrapper element (either <*xformsdb:insert*>, <*xformsdb:delete*>, or <*xformsdb:update*>, respectively) to ensure that an undesired operation is not executed by mistake. As a result of a successful submission, the XFormsDB implementation returns the metadata about files associated with the performed operation in the structure described in Section 5.9.

Table 15: Required attributes of the *xformsdb:file* element(s) for each operation associated with the *file* request

Operation	Required attributes
select	None. All files will be selected if filtering has not been used.

insert	*displayname*, *roles*, *filename*, *mediatype*, *filesize*, *comment*, and *creator*. Values for the *filename*, *mediatype*, and *filesize* attributes are automatically set by an *xforms:upload* element, which is bound to the aforementioned attributes.
delete	*id*
update	*displayname*, *roles*, *filename*, *mediatype*, *filesize*, *comment*, *creator*, *created*, *lastmodifier*, *lastmodified*, *id*, and *download*. Replacing values for the *filename*, *mediatype*, and *filesize* attributes are automatically set by an *xforms:upload* element, which is bound to the aforementioned attributes. Furthermore, replacing values for the *id*, *lastmodified*, and *download* attributes must be ignored by an XFormsDB implementation.

5.3.7 The *cookie* Request

The *cookie* request provides a means for checking browser support for cookies.

The *xformsdb:cookie* Element

Required child element of the *xformsdb:instance* element specifying only the name of the request.

5.4 The *xformsdb:submission* Element

The *xformsdb:submission* element is a new element that adds functionality to submit XFormsDB-related requests, such as the *query* requests. The requests can be submitted multiple times and at any point in the lifetime of a form.

The functionality and the processing model of the *xformsdb:submission* element is identical to the *xforms:submission* element. In addition, the element supports extension attributes, which have been described below.

Table 16: Extension attributes for the *xformsdb:submission* element

Attribute	**Description**
requestinstance	Required attribute specifying the *xformsdb:instance* element containing an XFormsDB-related request to be submitted. The default value is a reference to the first occurrence of the *xformsdb:instance* element.
statetype	Optional attribute specifying the type of the *state* request, whose legal values are: *get* (default) and *set*.
expressiontype	Optional attribute specifying the type of the *query* request, whose legal values are: *all* (default), *select*, and *update*.
filetype	Optional attribute specifying the type of the *file* request, whose legal values are: *select* (default), *update*, *insert*, and *delete*.
attachmentinstance	Optional attribute specifying the instance to be sent as an attachment along with: (1) the *state* request (*set*), (2) the *query* request (*update*), and (3) the *file* request (*update*, *insert*, and *delete*). The attachment instance contains the data to be updated, inserted, or deleted—depending on the request. When the attribute is absent, defaults to the value of the *instance* attribute.

5.5 The *xformsdb-request-error* Event

The *xformsdb-request-error* event is a new, notification-typed event that is dispatched as an indication of a failure of an XFormsDB-related request submission and/or execution process. For instance, if an error occurs in establishing a connection to a data source or in executing a query expression. The event can be caught by, similarly to other events, XForms event handlers (XForms Actions) that use the events system defined in DOM Level 2 Events [13] and XML Events [14].

The properties of the *xformsdb-request-error* event have the following values:

- Target element: *xformsdb:submission*
- Bubbles: Yes
- Cancelable: Yes
- Context info: None
- Default action: None; notification event only

In addition to the dispatched *xformsdb-request-error* event, a detailed error from an XFormsDB implementation is made available. Generally, the error is appended into the root node of the first child element of the *xformsdb:instance* element, which submitted the XFormsDB-related request. However, in case the *replace* attribute of the *xformsdb:submission* element has the value *all*, the error is included in an XHTML document returned after the submission to guarantee that the error can be displayed in XHTML browsers which do not necessarily support plain XML documents.

5.6 The *xformsdb:secview* Element

The *xformsdb:secview* element is a new element that enables showing/hiding a part of a Web page based on the roles of a user. For instance, a Web page may show a login form for users who have not logged in yet, whereas for logged-in users the

Web page may show a username and a logout button in the exact same area on the Web page.

Before an XFormsDB implementation sends a Web page to the client, it must go through all the *xformsdb:secview* elements on the Web page and check whether or not the current user has rights to access the content inside those particular *xformsdb:secview* elements. In case the current user meets all the conditions set, then, and only then, the content is exposed to the user.

In the absence of the underlying attributes, the content inside the particular *xformsdb:secview* element is shown only for users who have not logged in yet. A detailed decision tree diagram of the *xformsdb:secview* element is presented in Appendix B.

Table 17: Attributes of the *xformsdb:secview* element

Attribute	**Description**
roles	Optional attribute specifying the space-separated list of role names associated with the part of a Web page. A user must belong to any of the listed roles in order to access the content inside the particular *xformsdb:secview* element on the Web page.
allroles	Optional attribute specifying the space-separated list of role names associated with the part of a Web page. A user must belong to all of the listed roles in order to access the content inside the particular *xformsdb:secview* element on the Web page.
noroles	Optional attribute specifying the space-separated list of role names associated with the part of a Web page. A user must *not* belong to any of the listed roles in order to access the content inside the particular *xformsdb:secview* element on the Web page.

noallroles	Optional attribute specifying the space-separated list of role names associated with the part of a Web page. A user must *not* belong to all of the listed roles in order to access the content inside the particular *xformsdb:secview* element on the Web page.

5.7 The *xformsdb:include* Element

The *xformsdb:include* element is a new element that provides an inclusion mechanism to facilitate modularity. By means of the *xformsdb:include* element, it is possible to build large XML documents out of several well-formed XML documents. The idea behind the *xformsdb:include* element is similar to XInclude [102] with the difference that it is much simpler.

The processing of the *xformsdb:include* elements is recursive, i.e., an included XML document can itself include another XML document.

Table 18: Attributes of the *xformsdb:include* element

Attribute	**Description**
resource	Required attribute indicating the URI of an external XML document to be included. Behavior of relative URIs in links is determined by the host language, i.e., the form. Default value is an empty string.

5.8 The *xformsdb_users.xml* Document

The *xformsdb_users.xml* document is the data source (realm) of usernames and passwords that identify valid users of a Web application, plus an enumeration of the list of roles associated with each valid user. A particular user can have any number of roles associated with their username.

Updating, inserting, and deleting users is described in Section 5.3.5.

The *xformsdb:users* Element

Required element that wraps all *xformsdb:user* elements, i.e., it identifies valid users of a Web application.

The *xformsdb:user* Element

Required child element of the *xformsdb:users* element, which identifies a valid user of a Web application.

Table 19: Attributes of the *xformsdb:user* element

Attribute	Description
username	Required attribute specifying the username of a user, which is used for logging into a Web application. Each user must have a unique username within the Web application.
password	Required attribute specifying the password of a user, which is used for logging into a web application. A password can be either in clear text or hashed (supported cryptographic hashing algorithms: SHA-512, SHA-384, SHA-256, SHA-1, and MD-5; supported encoding methods: hex and base64).

roles	Required attribute specifying the space-separated list of role names associated with a user.
anyAttribute	Foreign attributes are allowed on this element.

5.9 The *xformsdb_files.xml* Document

The *xformsdb_files.xml* document contains the metadata about files associated with a Web application, such as display name and file size. The actual files, which are uploaded by the users of the Web application, on the other hand are stored either within the Web application or to another location on the server.

Updating, inserting, and deleting files is described in Section 5.3.6.

The *xformsdb:files* Element

Required element that wraps all *xformsdb:file* elements, i.e., it contains the metadata about files associated with a Web application.

The *xformsdb:file* Element

Required child element of the xformsdb:files element, which contains the metadata about a file associated with a Web application.

Table 20: Attributes of the *xformsdb:file* element

Attribute	Description
displayname	Required attribute specifying the display name of a file.
roles	Required attribute specifying the space-separated list of role names associated with a file.

filename	Required attribute specifying the name (set automatically by an *xforms:upload* element) of a file.
mediatype	Required attribute specifying the MIME type (set automatically by an *xforms:upload* element) of a file.
filesize	Required attribute specifying the size (set automatically by an *xforms:upload* element) of a file.
comment	Required attribute specifying the free form comment associated with a file.
creator	Required attribute specifying the creator (username) of a file.
created	Required attribute specifying the creation date (generated automatically by an XFormsDB implementation) of a file in the xs:dateTime format.
lastmodifier	Required attribute specifying the last modifier (username) of a file.
lastmodified	Required attribute specifying the last modified date (generated automatically by an XFormsDB implementation) of a file in the xs:dateTime format.
id	Required attribute specifying the ID of a file, which is used for locating files form the file system on the server. Each file must have a unique ID (generated automatically by an XFormsDB implementation) within a Web application.

download	Additional attribute pointing to the URI from which a file can be downloaded. The attribute is not stored in the *xformsdb:file* element but added automatically by an XFormsDB implementation when the metadata of the file is retrieved. In case a user does not have rights to download the file, the XFormsDB implementation must show an error message.
anyAttribute	Foreign attributes are allowed on this element.

5.10 Security Considerations

It must be noted that for security reasons, sensitive information (e.g., query expressions and data source configurations) should never be exposed to the client in their original form. An XFormsDB implementation must take care of this issue, for example, by replacing query expressions and data source configurations with opaque reference IDs in order to prevent malicious clients from rewriting the query expressions and stealing the data source configurations.

In addition, redirecting to a (or the same) Web page is in most cases required upon a successful login and logout in order to generate a new view with up-to-date access rights for a user.

Finally, a two second pause should be applied by an XFormsDB implementation after each attempt to download a file using an incorrect Uniform Resource Locator (URL) in order to secure file downloads against malicious clients.

5.11 Summary

In this Chapter, the requirements and design of the XFormsDB markup language have been presented. Finally, the XFormsDB markup language is evaluated with respect to the requirements, whose results are presented in Table 21.

Table 21: The XFormsDB markup language requirements and related work in this Thesis

Requirement	Related work in this Thesis	Section
LR1: The syntax and processing model of the XFormsDB markup language must be similar to XForms.	The syntax and processing model of the XFormsDB markup language resembles XForms.	5.2 – 5.10
LR2: The architecture of the XFormsDB markup language must be easily extensible.	New XFormsDB-related requests can be added to the language without requiring any changes to the request submission process.	5.3
LR3: XFormsDB-related requests must be able to be executed multiple times and at any point in the lifetime of a form.	The *xformsdb:submission* element addresses this requirement.	5.4
LR4: The XFormsDB markup language must provide a means for notifying XFormsDB-related request errors including detailed error messages.	The *xformsdb-request-error* event along with a detailed error message is made available upon an unsuccessful submission.	5.5
LR5: The XFormsDB markup language must provide a means for facilitating modularity in XHTML+XFormsDB documents.	The *xformsdb:include* element makes possible to build large XML documents out of several well-formed XML documents.	5.7

LR6: The XFormsDB markup language must provide a means for maintaining state in XFormsDB Web applications.	This requirement is addressed by the s*tate* request.	5.3.1
LR7: The XFormsDB markup language must provide a uniform API for connecting to different types of data sources.	The *datasrc* and *doc* attributes provide an abstraction layer on the language level.	5.3.2 and 5.3.5
LR8: The XFormsDB markup language must provide a mechanism for authentication and access control.	Authentication is addressed by the *login*, *logout*, and *user* requests, whereas access control is addressed by the *xformsdb:secview* element together with the *xformsdb_users.xml* document.	5.3.2, 5.3.3, 5.3.4, 5.6, and 5.8
LR9: The XFormsDB markup language must support a standardized query language applicable across different types of XML data sources.	The XFormsDB markup language supports both XQuery and XPath.	5.3.5
LR10: The XFormsDB markup language must provide a means for supporting multi-user concurrency.	Updating data to be stored in a data source can be done either *with* or *without* data synchronization.	5.3.5

LR11: Security issues must be considered carefully in the design of the XFormsDB markup language.	Security issues are addressed throughout the design of the XFormsDB markup language and summarized in the second-to-last Section of this Chapter.	5.10

Chapter 6

Implementation of the XFormsDB Framework

This Chapter describes the XFormsDB framework, which is a Proof-of-Concept (PoC) implementation to confirm the feasibility of the XFormsDB markup language. The Chapter starts by specifying the main requirements for the XFormsDB framework. Then, the architecture and features of the framework are described in detail. Finally, evaluation of the implemented framework with respect to the requirements is carried out.

6.1 Requirements

A detailed list of the main functional and non-functional requirements for the XFormsDB framework (hereinafter referred to as the Framework Requirements, FR) is presented in Table 22. The framework requirements were primarily derived from typical usage scenarios and technical specifications.

Table 22: Requirements for the XFormsDB framework

ID	Requirement
FR1	**The XFormsDB framework must implement the XFormsDB markup language.** The XFormsDB framework must support the XFormsDB markup language, i.e., offer all the features specified by the language. This is the main requirement for the framework.
FR2	**The XFormsDB framework must be able to support different types of user agents simultaneously.** Obviously, today's user agents are not yet able to interpret the syntax of the XFormsDB markup language, and thus are not capable of processing authored XHTML+XFormsDB documents as such. Therefore, the documents must be transformed into other formats (e.g., (X)HTML+CSS+JavaScript) viewable by different types of user agents.
FR3	**The XFormsDB framework must be able to support different types of data sources simultaneously.** Persistent data can be stored in various databases (e.g., native XML databases or relational databases) and formats (e.g., XML or relational data). Because of this, the XFormsDB framework must provide a means for XFormsDB Web applications to access various data sources.
FR4	**The architecture of the XFormsDB framework must be divided into logical tiers.** The user interface, business logic, and data must be separated. In addition, the framework must be build in a modular way, which allows replacing or adding components (modules) without affecting the rest of the system.

FR5	**The XFormsDB framework must support various Web standards and technologies.** In general, Web content authors use open standards (e.g., (X)HTML and CSS) in Web application development. Each standard has its own special purpose, for instance, (X)HTML is for document structure and CSS is for presentation. In order to utilize the pre-existing skill set of the bulk of Web content authors, the XFormsDB framework must not preclude the use of any open standard.
FR6	**The XFormsDB framework must provide transaction support and data synchronization capabilities.** Web applications are typically accessed concurrently by multiple users over a network, such as the Internet or an intranet. In general, when two or more users make updates to the same data fragment simultaneously, the updates made by the last user override the updates made by the previous users. In some cases, this is not acceptable, and therefore, the framework must perform data synchronization before committing an update transaction.
FR7	**The XFormsDB framework must be able to manage sessions between the client side and the server side regardless of the user agent used or its settings.** HTTP(S) is a stateless protocol, but Web applications often need to maintain session state for every user. Typically, this is achieved by using cookies—small pieces of data stored in user agents. In some cases, however, the use of cookies might not be possible. Therefore, in addition to cookies, an alternative way for managing sessions must be provided.

FR8	**The XFormsDB framework must be able to handle, report, and log errors.** Error handling is an essential part of every application. In case of an error (e.g., conflict in data synchronization), an appropriate error message must be sent to the user agent and the error must be logged for future reference.
FR9	**The XFormsDB framework must be highly customizable but yet easy to install and configure.** Different users have different needs and preferences. In addition, not all users possess advanced computer skills. Therefore, the installation and configuration of the framework must be as easy as possible but yet be flexible enough to suit for advanced users as well.

6.2 Development Environment

The run-time and development environment of The XFormsDB framework consists of several open-source application software and libraries. The main software used are:

Apache Ant 1.6.5 and Ant-Contrib 1.0b3 Apache Ant [67] is a Java-based build tool. It is a platform-independent replacement for the *make* tool used for automating build tasks, such as compiling Java classes and deploying Web applications. The Ant-Contrib project [80] extends the functionality of Apache Ant by providing a collection of additional tasks.

Apache Tomcat 5.5.27 Apache Tomcat [68] is a servlet container implementing the Java Servlet and JavaServer Pages (JSP) specifications from Sun Microsystems, Inc. In this project, Apache Tomcat is used as a Web server for hosting Java 2 Platform, Enterprise Edition (J2EE) Web applications.

Eclipse IDE for Java EE Developers 3.4.0 with Subclipse 1.2.4 Eclipse [81] is a collection of open source projects focused on building an open development platform comprised of extensible frameworks, tools, and runtimes for building, deploying, and managing software across the lifecycle. In this project, Eclipse IDE extended with the Subclipse plug-in [82] is used for developing J2EE Web applications.

eXist-db 1.2.4-rev8072-20080802 eXist-db [69] is a native XML database with broad support for standards, technologies, and APIs, including XQuery + update extensions, REST, and XML:DB API [75]. It is written in the Java language and it runs on most major platforms. In addition, eXist-db supports different alternatives for server deployment, ranging from a standalone server process to a Web application.

3DM 0.1.5beta1 3DM [70, 85] is a middleware for performing three-way merging and differencing of XML documents. The XFormsDB framework utilizes 3DM in updates that require data synchronization.

Orbeon Forms dev-post-3.7.1.200910160000-development[1] Orbeon Forms [72] is a J2EE based framework for building XML-centric Web applications. The XFormsDB framework utilizes AJAX-based Orbeon Forms XForms processor[2] in order to support standard Web browsers, including Internet Explorer, Firefox, and Safari.

In addition, Java Development Kit (JDK) 1.5.0 [71] or later is required.

6.3 High-Level Architecture

The high-level architecture of the XFormsDB framework is illustrated in Figure 9. As the figure shows, several XFormsDB Web applications can reside and run simultaneously on a single Web server without interfering each other. The Web

[1] This version contains important bug fixes necessary for the XFormsDB framework to work properly.
[2] In fact, XForms processing is shared between a light client-side module and a heavier server-side module that rely on the AJAX technique.

applications rely on a generic server-side component (XFormsDB processor), which interprets authored XHTML+XFormsDB documents and provides the integration services to heterogeneous data sources. In addition to the XFormsDB Web applications, the Web server also hosts a server-side implementation of XForms called Orbeon Forms, which contains an XForms processor and is running as a separate Web application on the Web server.

The benefits of integrating the Orbeon Forms XForms processor with XFormsDB Web applications using the separate deployment mode instead of the other option, the integrated deployment mode, are: (1) several XFormsDB Web applications can utilize the same instance of the Orbeon Forms XForms processor, (2) easier upgrades of both XFormsDB Web applications and Orbeon Forms, (3) prevents situations where different versions of Java Archive (JAR) files could conflict, and (4) cleaner application architecture, which allows changing Orbeon Forms to another XForms implementation more easily if needed. [73]

On the client side, the XFormsDB framework supports three main types of user agents[1]: user agents with XForms 1.1 support, user agents with AJAX support, and user agents with plain (X)HTML support. All the main types of user agents can be supported simultaneously by providing a dedicated version of a Web page for each type of user agent. The detection of the type of user agent (and possibly redirection) is performed on the client side.

For storing persistent data, two different types of XML-based data sources are supported: XML documents and eXist-db (NXD). In the case of XML documents, the XFormsDB framework uses Saxon's implementation of XQuery and XQuery API for Java (XQJ) for executing XQuery expressions, whereas in the case of eXist-db (NXD), XML:DB API (which is implemented over XML-RPC [76]) is used for talking to the remote database engine and executing XQuery expressions. By using, for instance, a middleware described in Section 3.3.5, support for other types of data sources (e.g., relational databases) could be easily added as well (cf. dashed lines in Figure 9 and Figure 10).

[1] User agents supporting XForms 1.1 can be further subdivided into those that support JavaScript and those that do not support JavaScript. In addition, the XFormsDB framework is also capable of serving XHTML+XFormsDB documents as such.

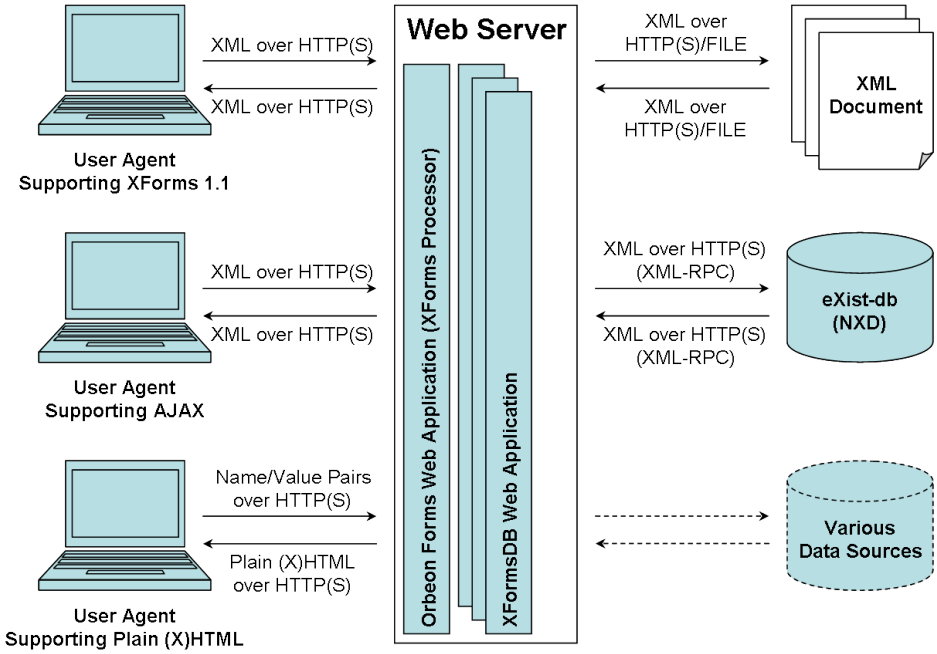

Figure 9: XFormsDB high-level architecture

Installation

Currently, the XFormsDB framework's one-click installer deploys eXist-db (NXD) as a separate Web application on the same Web server rather than as a standalone server, which makes the installation process of the XFormsDB framework as a whole much easier in a development environment. In a production environment, however, the standalone deployment is recommended, since it is more reliable and efficient than the Web application setup [77].

6.4 Modules and Tiers

The architecture of the XFormsDB framework (cf. Figure 10) can be divided into five logical tiers: client tier, presentation tier, application server tier, integration service tier, and data tier [83].

The *Client Tier* represents different types of user agents. Its purpose is to render the presentation prepared by the *Presentation Tier* as well as to react to the input from the user and relay it to the *Presentation Tier*.

The *Presentation Tier* is responsible for the preparation of the output to the *Client Tier*. It handles all incoming requests submitted by the user and transforms requested XHTML+XFormsDB documents into a dedicated version for each type of user agent (cf. Section 6.8).

The *Application Server Tier* is in charge of executing the business logic of the XFormsDB framework. It manages XFormsDB-related requests and their results as well as performs data synchronization (cf. Section 6.9).

The *Integration Service Tier* provides an interface for the *Application Server Tier* to perform the data access operations. It applies the logic needed to extract data from the *Data Tier* using the XQuery language. Communication with the *Data Tier* is accomplished with a standard API, such as XQJ.

The *Data Tier* stores application data in a persistent store, such as eXist-db (NXD).

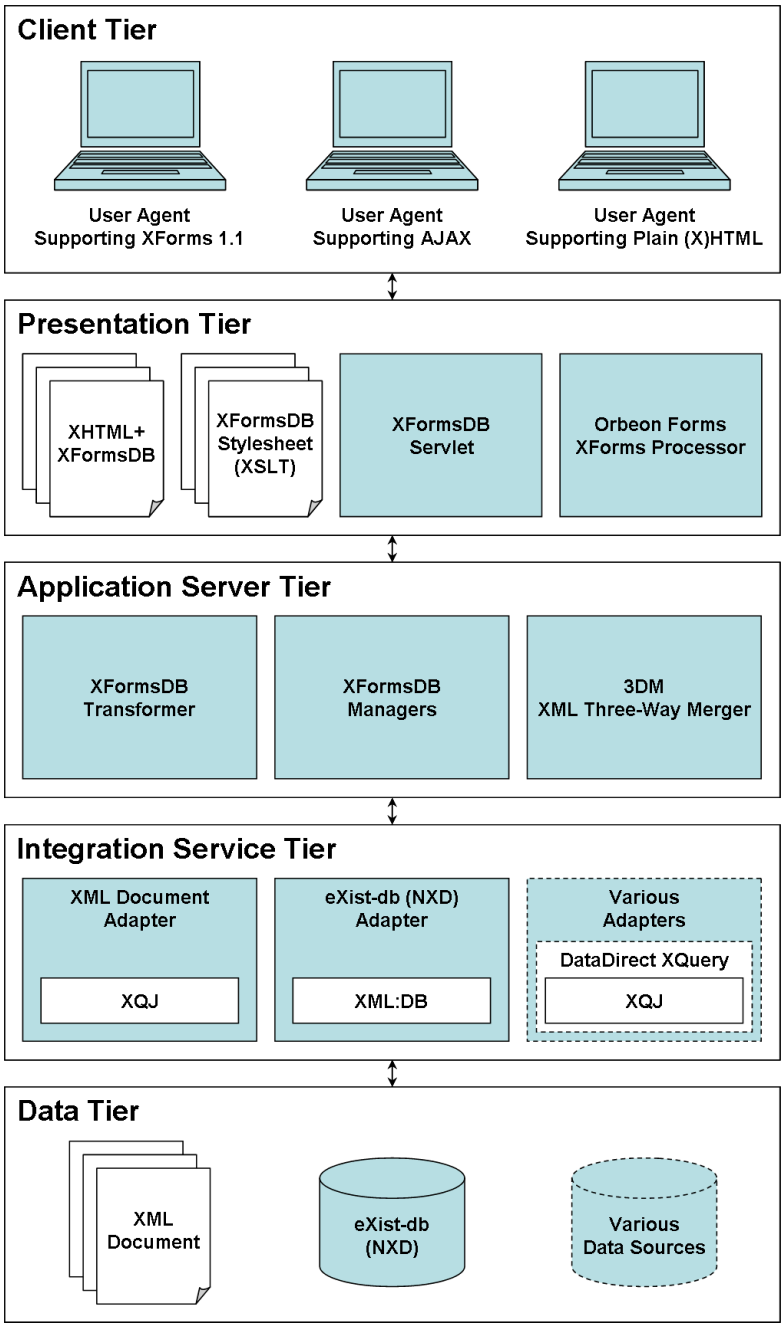

Figure 10: XFormsDB modules and tiers

6.5 Web Application Directory Structure

Figure 11 shows the directory structure of an XFormsDB Web application. It is based on the standard directory structure of a J2EE Web application because XFormsDB is a J2EE based framework.

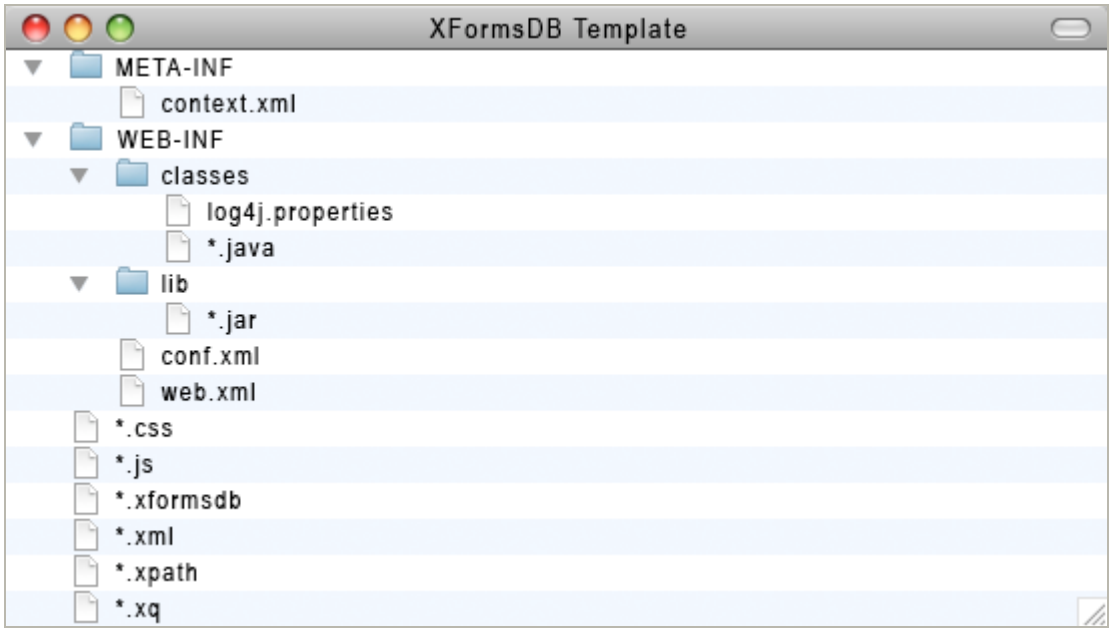

Figure 11: XFormsDB Web application directory structure

The *META-INF* directory contains the *context.xml* file, which allows forwarding requests to other Web applications, i.e., Orbeon Forms. The contents of this directory cannot be publicly accessed.

The *WEB-INF* directory contains the *web.xml* deployment descriptor file and the XFormsDB configuration file called *conf.xml* (cf. Section 6.12). The *classes* and *lib* subdirectories contain all the class, JAR, and resource files required by the Web application, including extension servlets written by the author of the Web application. The contents of this directory cannot be publicly accessed.

The root directory contains all the static files and additional subdirectories that make up the Web site.

Standalone versus Lite

There are two different deployment options for XFormsDB Web applications: *Standalone* and *Lite*. *Standalone* XFormsDB Web applications contain all the common class, JAR, and resource files required for running on any servlet container. *Lite* XFormsDB Web applications, however, do not contain these files, and therefore are significantly smaller in size and can only run on servlet containers supporting XFormsDB, i.e., servlet containers that make these files available to all deployed Web applications. As can be seen, both deployment options have benefits and drawbacks, and therefore usage must be assessed on a case by case basis.

6.6 Web Page Components

XFormsDB Web pages are authored using various Web standards and technologies, each with its own special purpose. The main Web standards and technologies used are:

- XHTML for document structure
- XForms for user interaction
- XFormsDB for data access and common server-side tasks
- XML for data model and interchange
- CSS for visual layout and presentation
- XQuery and XPath for querying data
- JavaScript for animation and additional user interaction

The relationships between the main Web standards and technologies used in XFormsDB Web pages are illustrated in Figure 12. The XHTML and XForms Web standards—including the XFormsDB technology—together form the base of a Web page. Other Web standards are included to the Web page either using inline style or adding a reference to an external file containing the definitions.

The benefits of making definitions into external files instead of using inline style are that it promotes component reuse and eases their maintenance. In addition, authoring of difficult components (e.g., complex query expressions) can be assigned to experts, after which those components can be easily shared with other non-expert authors.

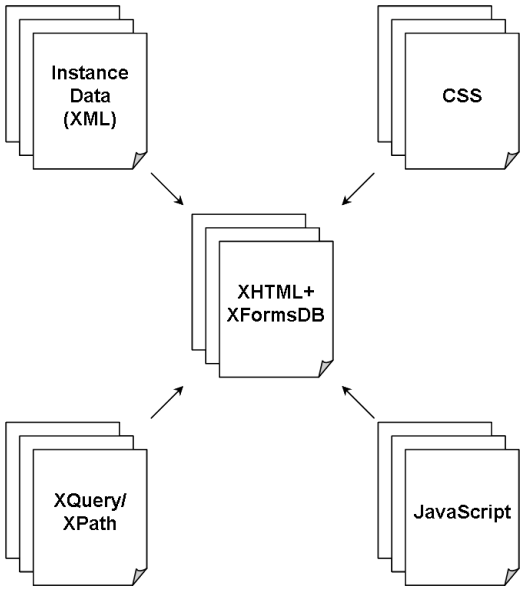

Figure 12: XFormsDB Web page main components

6.7 Handling Requests and Responses

In order to control and manage the handling of HTTP(S) requests in a centralized manner, the XFormsDB framework applies the Front Controller pattern (Servlet Front Strategy) [127]. The *XFormsDBServlet* class, which acts as the front controller, manages the handling of all requests, including authentication and authorization, delegating business processing, managing the choice of an appropriate view, and handling errors.

The process of handling a request is started by retrieving the HTTP(S) request parameters of interest (*id*, *expressiontype*, *statetype*, *filetype*, and *replacetype*). Then, the handling of the request is forwarded to an appropriate request handler based on the HTTP(S) request method (*get* or *post*), the HTTP(S) request content type (*null* or *application/xml*), and the file extension of the requested URL (**.xformsdb*,

**.xformsdbdownload*, **.xformsdbupload*, or other non-XFormsDB related file extension). Furthermore, if the HTTP(S) request method is *post*, then the content of the HTTP(S) request is analyzed as well. Finally, the response is written or, in case of a conditional *get* request that matches the condition, a *304 Not Modified* header is sent back to the client in the response.

6.8 Transformation Processes

The XFormsDB framework uses two separate server-side transformation processes (cf. Figure 13) to transform authored XHTML+XFormsDB documents (**.xformsdb*) into other formats viewable by different types of user agents.

The benefit of implementing both the XFormsDB language and the XForms language as separate server-side transformations, rather than implementing native support for the XFormsDB language in an open-source browser supporting XForms natively, such as X-Smiles browser [23, 24], is that the solution is not tied down to a single browser, but offers full cross-browser compatibility.

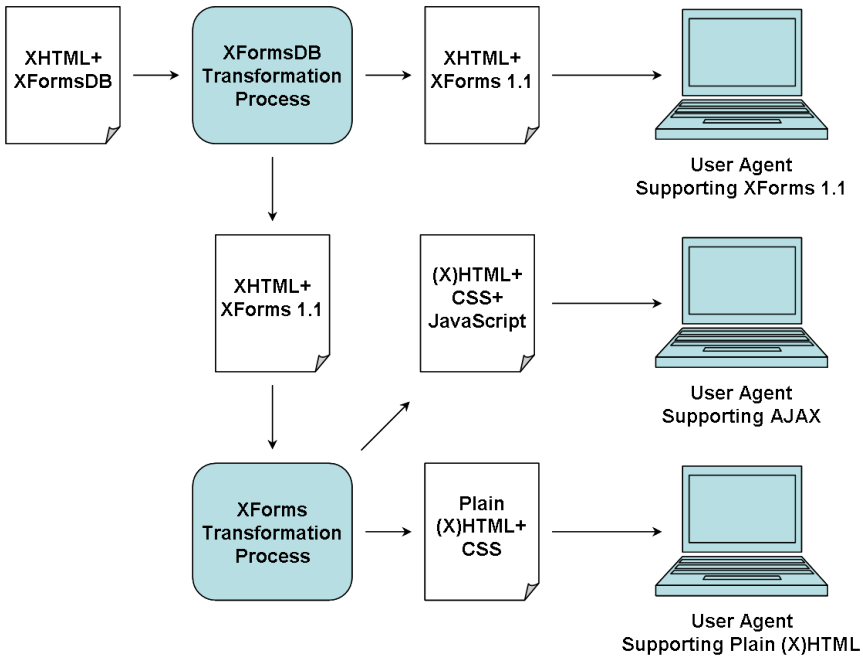

Figure 13: Transformation processes within the XFormsDB framework

6.8.1 XHTML+XFormsDB to XHTML+XForms 1.1

In the first transformation process, an authored XHTML+XFormsDB document is transformed into XHTML+XForms 1.1 compliant markup. The transformation process, which is performed by the *XFormsDBTransformer* class, is divided into seven main phases and it comprises four XSLT transformations altogether, as illustrated in Figure 14.

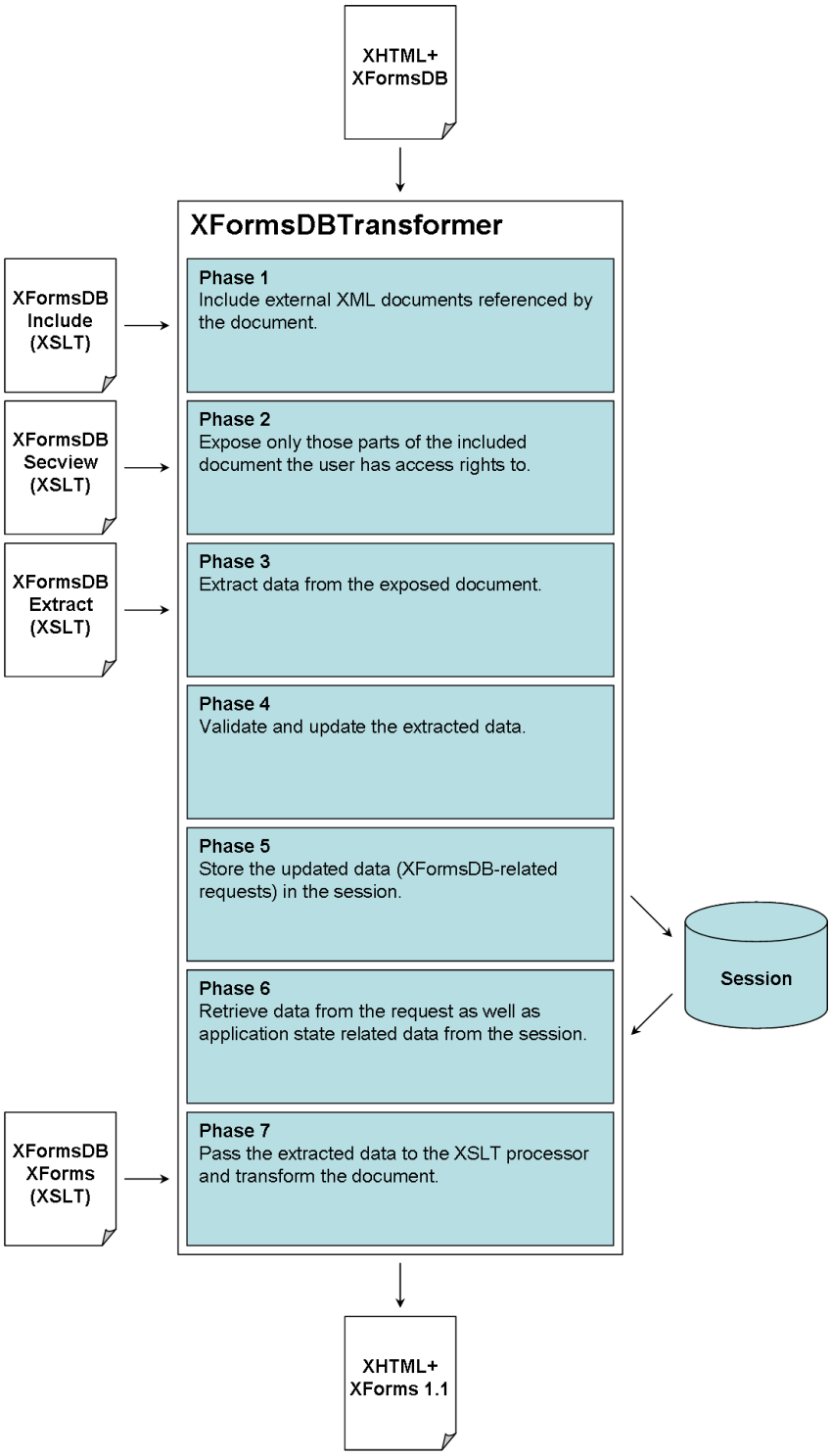

Figure 14: XFormsDB transformation process

In the first XSLT transformation (*xformsdb_include.xsl*), external XML documents (reusable XML fragments) are included in the main XHTML+XFormsDB document. The second XSLT transformation (*xformsdb_secview.xsl*) filters out those parts of the included document the user does not have access rights to, i.e., carries out authorization based on the roles of the user. The next XSLT transformation (*xformsdb_extract.xsl*) extracts data from the filtered document for validation and updating purposes. The updated data is then stored in the session for future reference. Finally in the last XSLT transformation (*xformsdb_xforms.xsl*), the filtered document is transformed into XHTML+XForms 1.1 compliant markup.

In addition, the following utility instances are automatically added to the document:

xformsdb-response-proxy-instance-x Acts as a response proxy for all the responses of XFormsDB-related requests. Added to each XForms model, in which "x" means the position of the XForms model within the document.

xformsdb-request-base-uri-instance Contains HTTP request base URI, for example, *http://localhost:8080/blog*. Added to the first XForms model only.

xformsdb-request-headers-instance Contains HTTP request headers. Added to the first XForms model only.

xformsdb-request-parameters-instance Contains HTTP request parameters, i.e., URL parameters. Added to the first XForms model only.

xformsdb-state-instance Contains Web application's state information, i.e., XFormsDB state. Added to the first XForms model only.

6.8.2 XHTML+XForms 1.1 to (X)HTML+CSS+JavaScript or Plain (X)HTML+CSS

In the second transformation process, the output of the previous transformation process, i.e., XHTML+XForms 1.1 compliant markup, is transformed into (X)HTML+CSS+JavaScript or plain (X)HTML+CSS, depending on the configuration. The transformation process is performed on the server side using a

third-party software, an AJAX-based XForms implementation called Orbeon Forms XForms processor (cf. OPS XForms Engine in Figure 15).

The transformation process as well as its position between the XFormsDB framework (XFormsDB Web applications) and user agents is illustrated and described in detail in Figure 15.

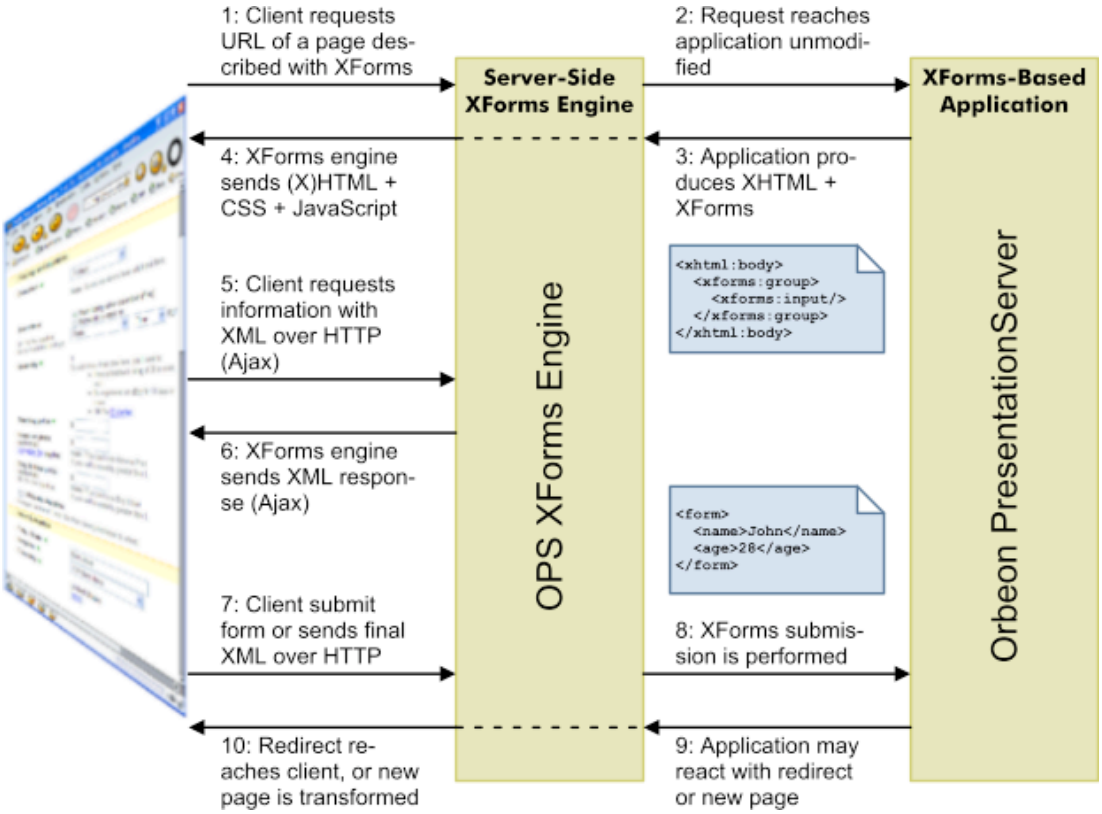

Figure 15: XForms transformation process [74]

The rationale behind plain (X)HTML+CSS support is to allow targeting user agents that either do not support JavaScript or have JavaScript disabled, such as old browsers and low-end mobile phones.

6.9 Data Synchronization

The XFormsDB framework includes built-in support for performing synchronized updates as specified in Section 5.3.5. To accomplish data synchronization, the XFormsDB framework uses 3DM [85], a middleware for performing three-way merging of XML documents, which is able to detect and handle *update*, *insert*, and *delete* operations as well as *moves* and *copies* of entire subtrees. Furthermore, the aforementioned operations can be performed without the use of unique element identifiers, i.e., XML documents can be used as such.

To illustrate how 3DM works, consider the merging example shown in Figure 16. In the example, T_0 is referred to as the original version, T_1 as the altered version, T_2 as the current version stored in the data source, and T_m as the merged version. Blue color indicates that the node has been either updated (marked with an apostrophe), inserted, or moved, whereas white color indicates that the node has remained unaltered.

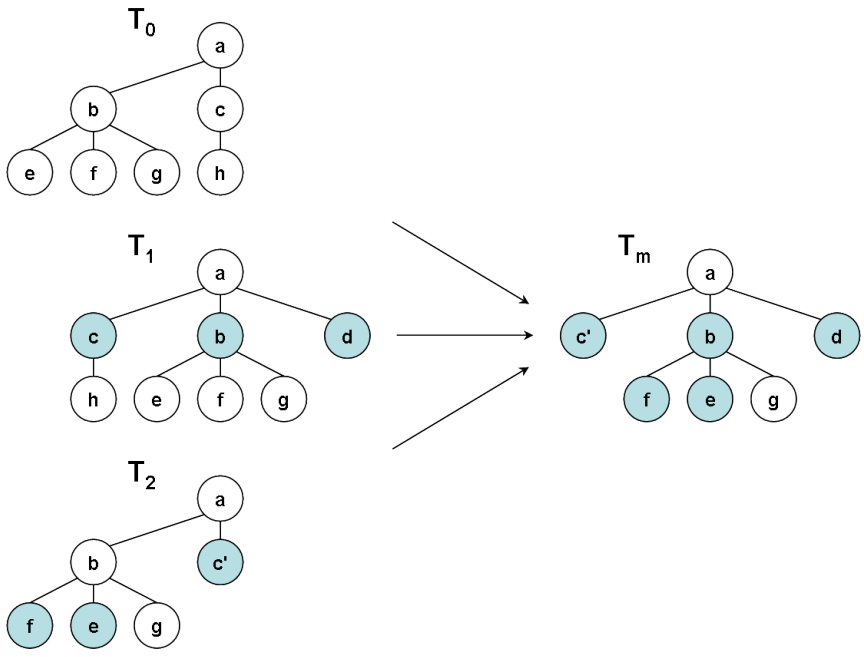

Figure 16: Data synchronization process: a three-way merge for XML documents

In case the data synchronization process fails (e.g., merge conflict), an appropriate error message is reported back to the form, which handles the error on a case by case basis.

Even though the solution for performing synchronized updates in the XFormsDB framework has many advantages (e.g., simple and elegant), it has a few disadvantages that need to be taken into account. For example, in some cases large XML fragments need to be transmitted back and forth just to perform a simple *insert* or *delete* operation. In addition, an XML fragment that needs to be updated might expose sensitive information to the client side.

6.10 Session Management

The stateless nature of HTTP(S) forces developers to find other ways for managing sessions between the client side and the server side. The most popular way is through the use of session identifiers, in which a session identifier (a unique session ID) is transmitted back to the server with every HTTP(S) request.

There are three ways available to both allocate and receive session ID information, each having its advantages and disadvantages [84]:

- Cookies
- URL rewriting
- Hidden form variables

The XFormsDB framework supports two out of the three ways for managing sessions: cookies (default) and URL rewriting (can also be used as default).

The rationale behind URL rewriting support is to allow targeting user agents that either do not support cookies or have cookies disabled, such as low-end mobile phones. In addition, URL rewriting allows multiple simultaneous sessions for a single user, i.e., it makes possible running two or more instances of the same Web application and/or widget within the same user agent instance.

6.11 Error Handling

The XFormsDB framework implements error handling as specified in Section 5.5. In case of an error, an appropriate error message with error code and error description is sent to the user agent. The format of the error message is either XML (cf. Listing 39) or XHTML (cf. Listing 40), depending on the request.

The error descriptions sent by the XFormsDB framework are kept a bit vague on purpose due to security reasons. Furthermore, they are not internationalized (in English only) because the XFormsDB framework is a general-purpose software. Therefore, XFormsDB Web applications usually display own specific, and possibly internationalized, error descriptions along with the sent error code when an error occurs.

In addition to the sent error message, the following information about the occurred error is written to a log file on the server in order to trace the problem to its source and to determine why the error occurred:

- Date and time
- Error code
- Error description
- Full Java exception stack trace
- Logged in user
- HTTP(S) request URL
- HTTP(S) request headers of the latest (*get* or *post*) request
- HTTP(S) request headers of the latest *get* request

6.12 Configuration

The main configuration file for the XFormsDB framework is called *conf.xml*, which is loaded from the *WEB-INF* directory of the XFormsDB Web application. Table 23 describes how the system can be configured and lists all the customizable settings.

Table 23: Settings of the XFormsDB configuration file (*conf.xml*)

Setting	Description
MIME mapping	The MIME mappings (extension and MIME type) that are used to transform XHTML+XFormsDB documents into XHTML+XForms 1.1 compliant markup.
Encoding	The character encoding that is used in input (e.g., query files) and output files throughout the XFormsDB Web application.
Data source	The predefined data source configurations that are used for connecting to the data source.
Files metadata	The predefined data source configuration that is used for connecting to the files metadata data source.
Files folder	The files folder that is used for storing uploaded files.
3DM conflict level	The conflict level of the three-way XML merging tool that is used for updating data (an XML fragment) with data synchronization.
Security file	The security files (extension) that are used for protecting files from clients.

The *web.xml* deployment descriptor file, which is also loaded from the same directory, contains the rest of the XFormsDB Web application settings, such as Orbeon Forms XForms processor related settings, session timeout, MIME mappings, and welcome files.

All of the settings in both configuration files default to reasonable values, thus making the XFormsDB framework ready to work "out-of-the-box".

6.13 Summary

This Chapter presented the requirements and implementation of the XFormsDB framework, including the architecture and features of the framework. Finally, the XFormsDB framework is evaluated with respect to the requirements, whose results are presented in Table 24.

Table 24: The XFormsDB framework requirements and related work in this Thesis

Requirement	Related work in this Thesis	Section
FR1: The XFormsDB framework must implement the XFormsDB markup language.	The XFormsDB framework conforms to the XFormsDB markup language.	6.3 – 6.12
FR2: The XFormsDB framework must be able to support different types of user agents simultaneously.	User agents supporting XForms 1.1, AJAX, and plain (X)HTML can be used on the client tier.	6.3, 6.4, and 6.8
FR3: The XFormsDB framework must be able to support different types of data sources simultaneously.	Currently, XML documents and eXist-db (NXD) are supported for storing persistent data.	6.3 and 6.4

FR4: The architecture of the XFormsDB framework must be divided into logical tiers.	The XFormsDB framework is based on a modular architecture, which separates presentation, business logic, and data.	6.3 and 6.4
FR5: The XFormsDB framework must support various Web standards and technologies.	The XFormsDB framework does not preclude the use of any open standard.	6.5 and 6.6
FR6: The XFormsDB framework must provide transaction support and data synchronization capabilities.	The XFormsDB framework includes a built-in support for performing synchronized updates.	6.9
FR7: The XFormsDB framework must be able to manage sessions between the client side and the server side regardless of the user agent used or its settings.	Cookies and URL rewriting are supported for managing sessions.	6.10
FR8: The XFormsDB framework must be able to handle, report, and log errors.	In case of an error, an appropriate error message with error code and error description is sent to the user agent. In addition, detailed information about the occurred error is written to a log file on the server.	6.11

FR9: The XFormsDB framework must be highly customizable but yet easy to install and configure.	The configuration settings are extensive and default to reasonable values. The XFormsDB framework works "out-of-the-box" and can be installed by running a single install script.	6.3 and 6.12

Chapter 7

Sample Web Applications

This Chapter describes two sample Web applications in detail, which were developed to validate the feasibility of the XFormsDB framework presented in Chapter 6 as well as to answer the research question *Q3* presented in Chapter 4.

The first sample Web application, PIM: Contacts, is a toy example motivated by reality and the second sample Web application, Blog, is a real-life Web application of today. Both of these two example types as validation techniques are also mentioned by Shaw [10].

7.1 About Measurements

The following metrics were measured for both Web applications in order to determine the amount of work required to develop each Web application and the performance of each Web application: component metrics, response size metrics, and response time metrics.

The component metrics were measured so that the files of each Web application were formatted in a way that each line contained only one piece of a component (e.g., element's start or end tag, content, or comment), after which the number of lines, elements, attributes, and rules used were calculated separately for each component.

The component metrics, however, do not tell the truth about the performance of a Web application. Therefore, two accurate, state-of-the-art tools were used for measuring performance related metrics of each Web application: (1) Charles [103], a web debugging proxy application and (2) Episodes [104, 105], a web performance measurement framework. Charles was used to measure the response size metrics and to throttle bandwidth, whereas Episodes was used to measure the response time metrics. The response time metrics were measured by calculating the average Web page load time of ten tests.

The performance measurements were made over a simulated DSL connection (1000 kbps/1000 kbps, round-trip latency 40 ms) using Firefox 2.0.0.20 running on iMac8.1 2.4 GHz Intel Core 2 Duo with 4 GB RAM client machine. The server machine, on which the Web applications were running, was configured to compress (gzip [106]) all text responses, including JavaScript and CSS files.

7.2 Personal Information Management (PIM): Contacts

7.2.1 Overview

Personal Information Management (PIM): Contacts is a simple Web application for storing, browsing, and managing information about your personal contacts, such as names, addresses, phone numbers, and e-mail addresses.

PIM was developed as a toy example, although inspired by real-life systems, to test capabilities and ease of use of the XFormsDB framework as well as how easily certain user interface functions (e.g., sorting data and changing the language of the user interface) can be authored by using the framework.

7.2.2 Conceptual Web Site Diagram

The structure of the PIM Web site is illustrated in Figure 17. The Web site comprises of only a single Web page (Contacts; Home page) that contains a list of contacts and

actions (A.1–A.4 and C.1–C.2), primarily for managing the contacts. Each contact within the Web page can be set to an individual state (B.1–B.4) without reloading the whole Web page.

The goal of having only a single Web page with multiple contact states, instead of having separate Web pages for each contact state, is to improve the overall user experience and usability of the Web site, i.e., by allowing users to manage their contacts without interruptions.

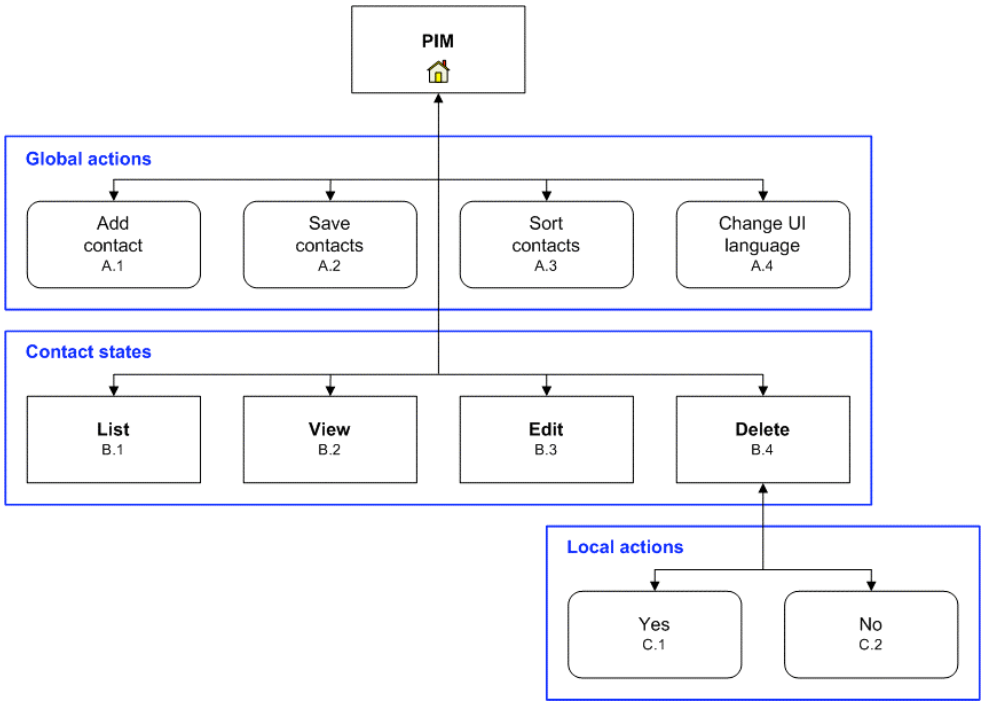

Figure 17: PIM conceptual Web site diagram

7.2.3 User Interface

Figure 18 shows the user interface of the PIM Web application in its initial state. In this state, all contacts are in the list state and sorted by name in ascending order. The default language of the user interface is English.

The language menu is located in the top right corner, which provides three different options for the user interface language: Finnish, Swedish, and English.

All contacts are listed in the *Contacts* table, which is located in the middle section of the Web page. Detailed information about a certain contact can be viewed by clicking the name of the contact. Contact information can be modified or removed by clicking the *Edit* or *Delete* link, respectively. By clicking the arrow icon next to the *Name* column, contacts can be sorted by name either in ascending or descending order depending on the previous state. The total number of contacts is displayed in the bottom left corner of the *Contacts* table.

Figure 19 shows the user interface after some of the contact information have been modified and the changes have been successfully saved to the database.

Finally, it should be noted that all the aforementioned actions can be carried out without reloading the whole Web page.

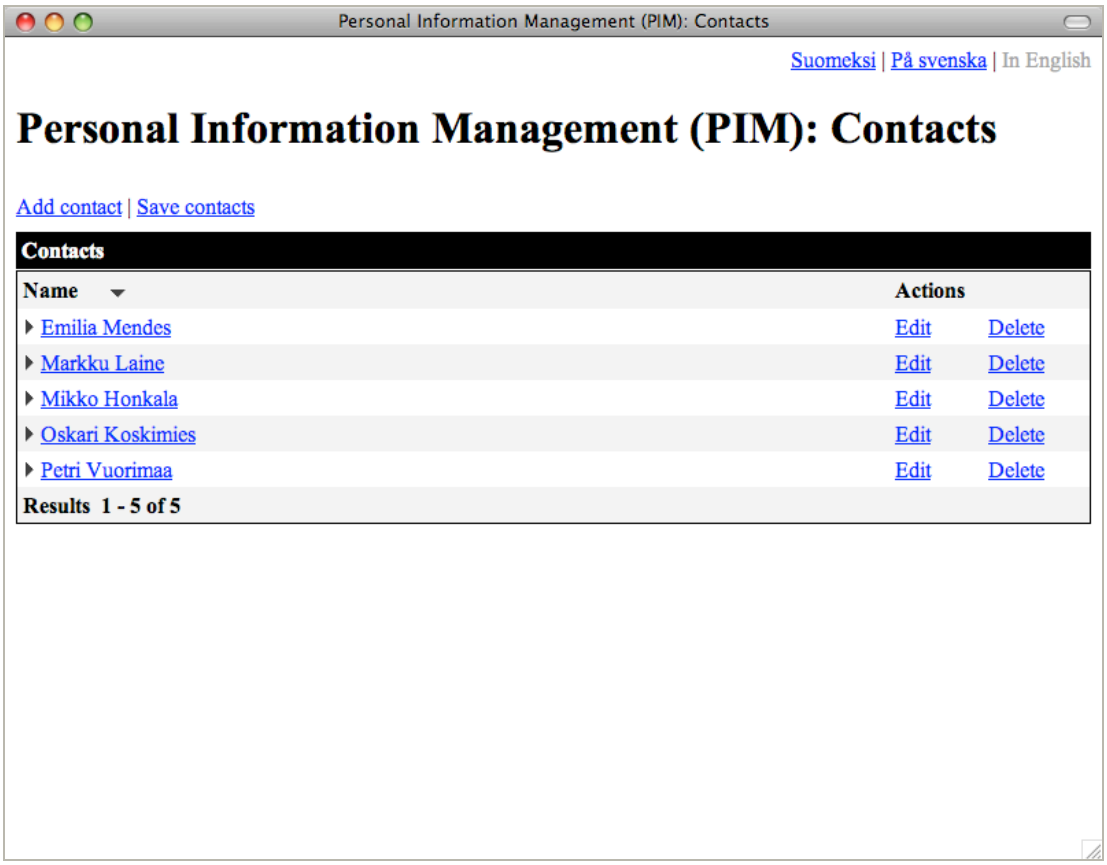

Figure 18: The user interface of the PIM Web application in the initial state, in which all contacts are in the list state

Figure 19: The user interface of the PIM Web application, in which contacts are in different states

7.2.4 Architecture

The PIM Web application was developed by using the XFormsDB framework. The framework was configured so that it makes use of both the mandatory XFormsDB

transformation process and the optional AJAX-based Orbeon Forms XForms Engine in order to transform the Web page it consists of into the form of (X)HTML+JavaScript+CSS, which is viewable by most common browsers. For storing persistent data, eXist-db (NXD) was configured to be used.

The Web page, from which the whole PIM Web application basically consists of, was authored purely declaratively[1] using the XHTML+XFormsDB markup languages. In addition to the standard functions provided by XForms, the Web page uses a few XForms extension functions (e.g., for data sorting) implemented by Orbeon Forms. The visual appearance of the user interface was tweaked by using external CSS files. Data templates were modeled in external XML files. Finally, the data query used within the PIM Web application was written in XPath and stored to a secured external file.

7.2.5 Queries

The PIM Web application contains only one simple query, which is aimed both for retrieving PIM data from and updating PIM data into eXist-db as a whole. The query relies solely on the features defined in the XPath 2.0 specification. The properties of the query are described in detail in Table 25.

Table 25: PIM queries

Property	Value
Name	select_and_update_pim.xpath
Description	Select and update PIM data.
Type and complexity	XPath, simple
External variables	no
Type of data source	eXist-db (NXD), remote
Result set size (XML)	medium

[1] Excluding the JavaScript code that was needed to instrument the Web page with Episodes timers.

7.2.6 XML Data

Listing 13 shows a snippet of the example XML document (PIM data) stored in the database. The data in the XML document has a regular structure (data-centric) and it mainly consists of a sequence of *contact* elements, each identified with a unique ID.

Listing 13: A snippet of the example XML document used in PIM

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <pim>
    <contacts>
      <contact id="faadfab7b61a0781c12374202c9765c3">
        <name>Markku Laine</name>
        <title>Research assistant</title>
        <company>Helsinki University of Technology</company>
        <street>Konemiehentie 2</street>
        <city>Espoo</city>
        <state></state>
        <zip>02150</zip>
        <country>Finland</country>
        <telephone></telephone>
        <fax></fax>
        <mobilephone>+358 50 555 3333</mobilephone>
        <email>markku.laine@tml.hut.fi</email>
      </contact>
      ...
    </contacts>
  </pim>
</root>
```

7.2.7 Metrics

The component metrics of the PIM Web application are presented in Table 26. As can be seen from the results, the number of lines that was needed for modeling the database template and connecting XForms to the database was extremely low, whereas the number of lines that was needed for authoring the structure, layout, and logic of the Web application was rather high. In addition, it must be noted that the

code was by no means optimized and it included a vast repetitive section of code (over 300 lines) due to alternating table rows.

Table 26: PIM component metrics

Component	Lines	Elements	Attributes	Rules
XHTML	386	202	196	0
CSS	194	0	0	33
JavaScript	30	0	0	0
XForms	443	329	390	0
Instance Data	143	132	27	0
XFormsDB	9	5	14	0
XPath	1	0	0	0
Database Template	6	3	0	0
Comment	93	0	0	0
Empty	30	0	0	0
Miscellaneous	12	3	2	0
Total	**1347**	**674**	**629**	**33**

Table 27 and Table 28 show the performance metrics of the PIM Web application. The measurements were made with both empty cache and primed cache. The total Web page weight was 92% less and the average Web page load time was 44% faster on subsequent page views.

Table 27: PIM response size metrics

	Empty Cache		Primed Cache	
	Requests	**Size (kB)**	**Requests**	**Size (kB)**
index.xformsdb				
HTML	1	12.2	1	12.2
JavaScript	3	111.1	2	0.0
CSS	3	6.4	2	0.0
Image	10	24.7	7	0.8
Total	**17**	**154.3**	**12**	**13.0**

Table 28: PIM response time metrics

	Empty Cache	Primed Cache
	Time (s)	**Time (s)**
index.xformsdb		
Backend	1.6	0.8
Frontend	0.6	0.5
Total	**2.2**	**1.3**

The sizes of the Web Archive (WAR) files were 37.9 kB (Lite version) and 17.6 MB (Standalone version).

7.2.8 Analysis

PIM toy example clearly demonstrated the feasibility and capabilities of the XFormsDB framework by proving that common Web application functions (e.g., internationalization, sorting data, and database access) can be authored purely

declaratively—does not require users to write any client-side scripting or server-side programming code at all.

Overall, developing the PIM Web application went easily without facing any serious problems. Internationalization of the user interface was accomplished fairly simply and rapidly by using conditional XPath expressions in XForms output form controls. In addition, another task that turned out extremely well was connecting XForms to the database, which was realized by utilizing XFormsDB features.

However, regardless of the success of the PIM Web application, the development process raised several issues that could be improved. Firstly, the current XForms standard lacks of many useful functions, such as sorting node sets. These functions are usually implemented as extensions by XForms implementations but they should be added to the XForms standard to ensure a unified authoring syntax. Secondly, both XHTML and XForms should provide support for Attribute Value Templates (AVT), i.e., allowing the inclusion of XPath expressions within attributes. This way the amount of repetitive code could be reduced significantly as there would not be the need for defining separate sections for each condition anymore. Finally, the update-typed *query* request of XFormsDB should be enhanced. At the moment, an XPath expression used for updating must point to the root element of the XML fragment to be updated which might result in large XML fragments even if only few nodes needs to be updated.

7.3 Blog

7.3.1 Overview

Blog is an online journal or diary Web tool for publishing personal contents, such as news, thoughts, comments, and experiences. It is a slightly simplified version of publicly available blog software.

Blog was developed to test how well the XFormsDB framework suits for authoring popular real-life Web applications of today, containing multiple Web pages and

complex data source queries. Mobile and widget versions of the Web application also exist but they have been excluded from the description.

7.3.2 Conceptual Web Site Diagram

The structure of the Blog Web site is illustrated in Figure 20. The Web site has been divided into two main areas: Public (B.1) and Administration (B.2).

In the Public area, users are allowed to browse through archives and read published posts as well as leave their comments on the posts. The Administration area, on the other hand, is controlled by limited access policy via the Login Web page (A.1). The area contains necessary tools for managing published posts and comments. Both the Public area and the Administration area consist of multiple page states (C.1–C.2 and C.3–C.5).

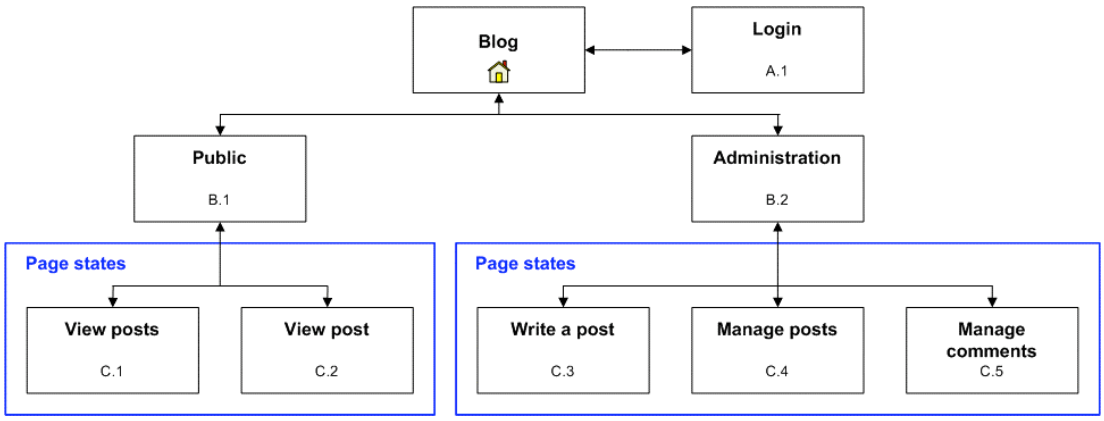

Figure 20: Blog conceptual Web site diagram

7.3.3 User Interface: Public

The user interface of the Public area is shown in Figure 21. The navigation takes place through the archive menu located on the right-hand side, which is created dynamically to list all months containing posts. The content part on the left-hand side is also created dynamically according to selected archive month and post.

The Web page by itself contains two page states in which the navigation between the states is carried out without reloading the whole Web page. For instance, Figure 22 shows the Public area in the view post state, which displays a single post with comments on it as well as the form for adding new comments.

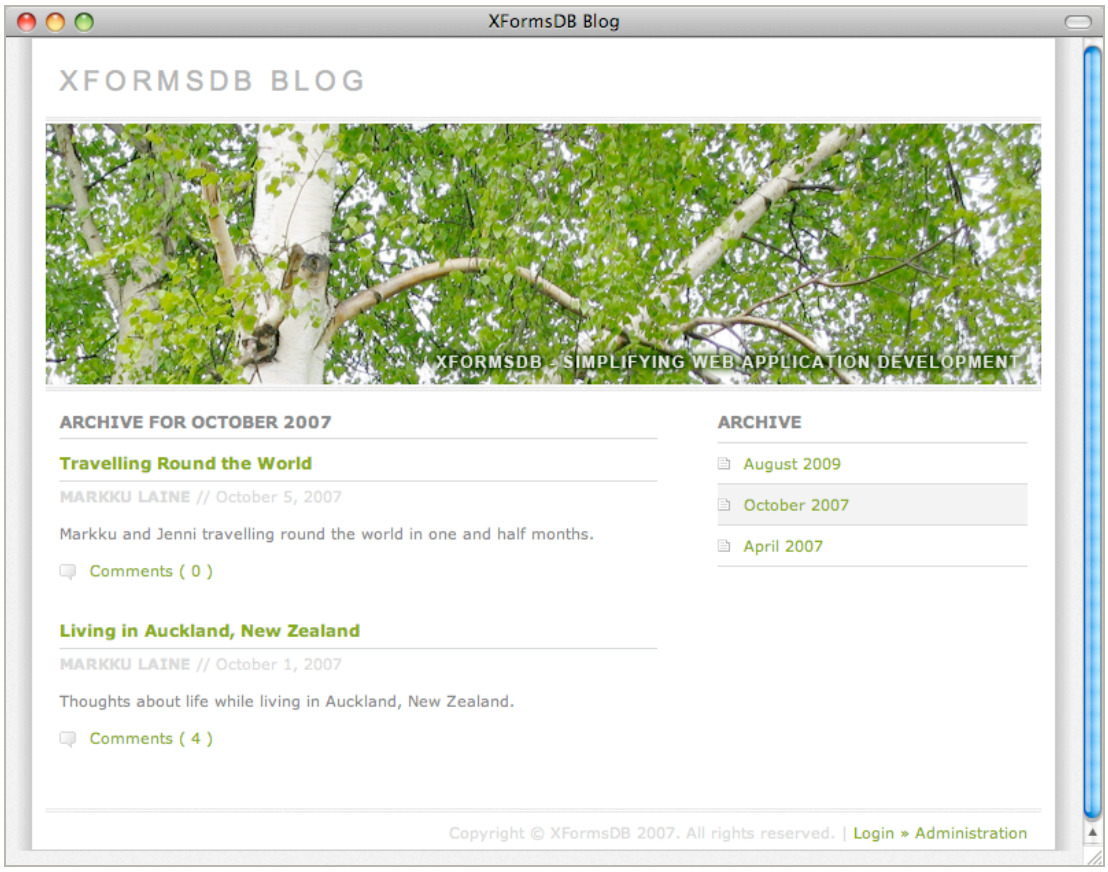

Figure 21: The user interface of the Public area of the Blog Web application in the view posts of the month state

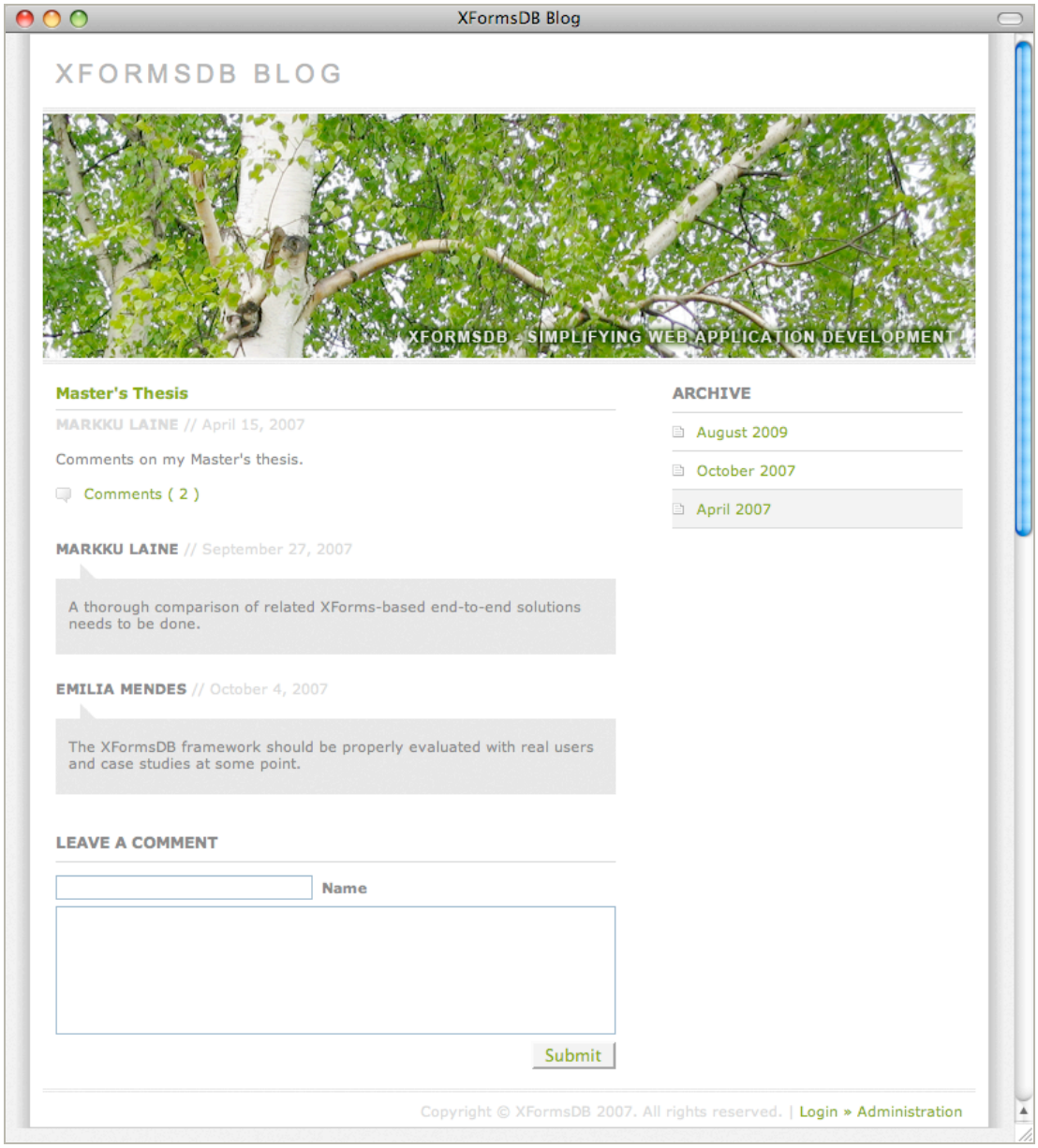

Figure 22: The user interface of the Public area of the Blog Web application in the view post state

7.3.4 User Interface: Administration

The user interface of the Administration area follows the same layout principles as the Public area which thus eases navigation through the Web site. The navigation, which is located on the right-hand side, has been divided according to the three main

tasks: write a post, manage posts, and manage comments. Depending on the selected task, appropriate tools for browsing, viewing, adding, editing, and deleting posts or comments are provided in the content part.

Figure 23 shows how comments can be smoothly managed, even without using a full-page refresh, using the tools in the Administration area.

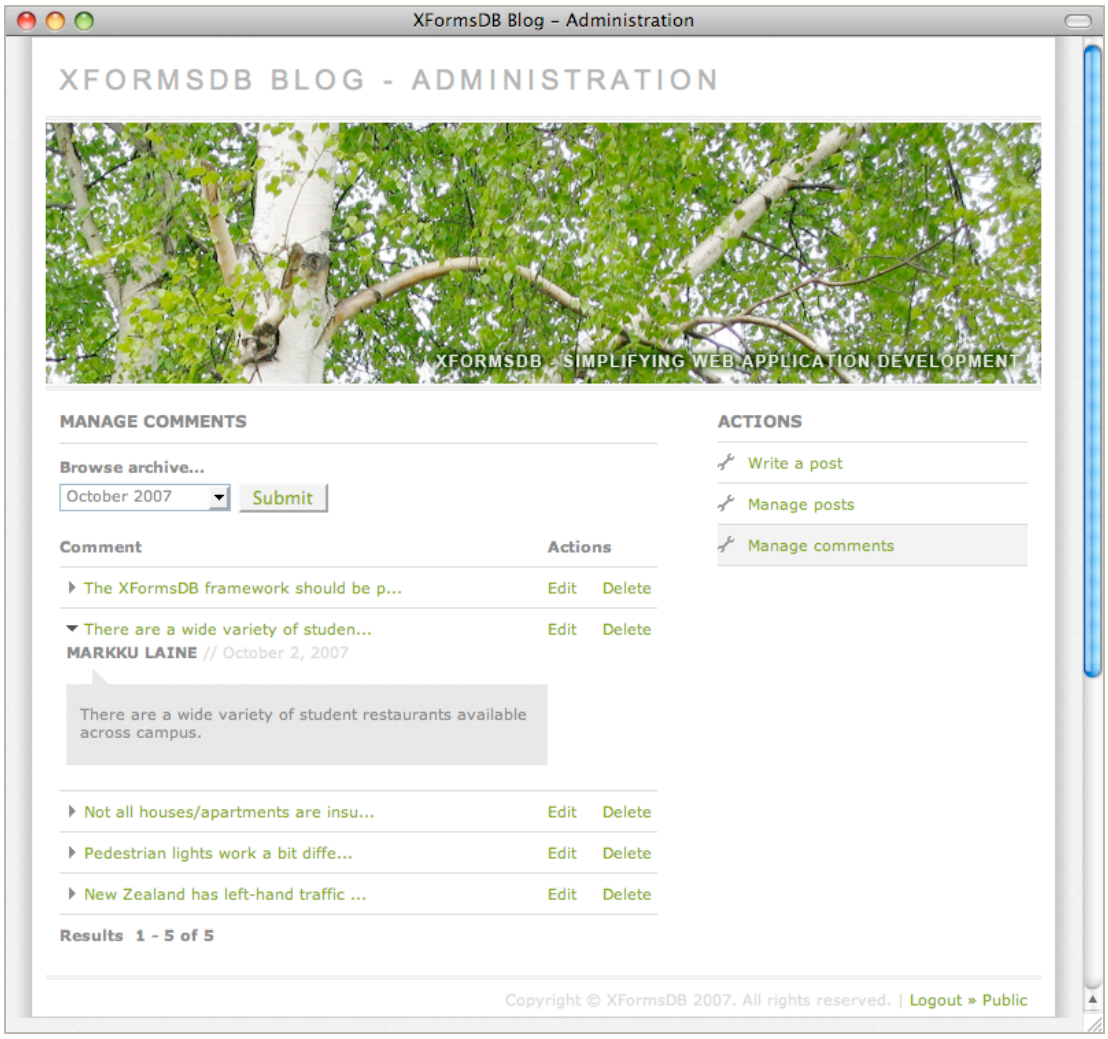

Figure 23: The user interface of the Administration area of the Blog Web application in the manage comments state

7.3.5 Architecture

The Blog Web application was developed using the XFormsDB framework with eXist-db (NXD) as the backend data source for the application. In addition, AJAX-based Orbeon Forms XForms Engine was used to provide support for browsers without XForms 1.1 support.

The Web pages of the Blog Web application were authored using the following Web standards and technologies: XHTML, XForms with Orbeon Forms extensions, XFormsDB, XML, XQuery, XPath, CSS, and JavaScript[1]. XML, XQuery, and CSS definitions were placed each in a separate file to maximize maintainability and reusability of components. In addition, configurable security files were used to protect sensitive information contained in XQuery and XPath files from malicious clients.

7.3.6 Queries

The queries used within the Blog Web application vary from simple XPath-based queries to complex XQuery-based queries utilizing external variables and functions defined in the FunctX XQuery Function Library [79]. The primary purpose of use of the queries is for selecting and updating posts and comments stored in eXist-db. Table 29 describes the properties of the queries used in detail.

[1] JavaScript was used only for instrumenting the Web pages with Episodes timers.

Table 29: Blog queries

Property	Value
Name	select_and_update_comments.xpath
Description	Select and update comments of a post.
Type and complexity	XPath, simple
External variables	yes
Type of data source	eXist-db (NXD), remote
Result set size (XML)	medium
Name	select_and_update_posts.xpath
Description	Select and update posts.
Type and complexity	XPath, simple
External variables	no
Type of data source	eXist (NXD), remote
Result set size (XML)	large
Name	select_comment_archives.xq
Description	Create a monthly list of archived comments.
Type and complexity	XQuery, complex
External variables	no
Type of data source	eXist-db (NXD), remote
Result set size (XML)	small
Name	select_comment.xq
Description	Select a comment of a post.
Type and complexity	XQuery, intermediate
External variables	yes
Type of data source	eXist-db (NXD), remote
Result set size (XML)	small

Name	select_manage_comments.xq
Description	Select filtered comments of all posts of a month.
Type and complexity	XQuery, complex
External variables	yes
Type of data source	eXist-db (NXD), remote
Result set size (XML)	medium

Name	select_manage_posts.xq
Description	Select filtered posts of a month.
Type and complexity	XQuery, complex
External variables	yes
Type of data source	eXist-db (NXD), remote
Result set size (XML)	medium

Name	select_post_archives.xq
Description	Create a monthly list of archived posts.
Type and complexity	XQuery, complex
External variables	no
Type of data source	eXist-db (NXD), remote
Result set size (XML)	small

Name	select_post.xq
Description	Select a post.
Type and complexity	XQuery, intermediate
External variables	yes
Type of data source	eXist-db (NXD), remote
Result set size (XML)	medium

Name	select_posts.xq
Description	Select posts of a month.
Type and complexity	XQuery, complex
External variables	yes
Type of data source	eXist-db (NXD), remote
Result set size (XML)	medium

7.3.7 XML Data

The Blog Web application uses two XML documents, *blog.xml* (cf. Listing 14) and *xformsdb_users.xml*, to store all the data needed. Spreading the data across more XML documents would also have been an option, but it would have resulted in considerably more complex queries.

The XML documents were designed with data-centric XML in mind, because the XML documents were meant for machine consumption only. Each *post* and *comment* element within *blog.xml* contains a required attribute called *id*, which identifies the element in question.

Listing 14: A snippet of the example XML document used in Blog

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <blog>
    <posts>
      <post id="08f865da2048e660daf7dbac35d7ba2a">
        <headline>Master's Thesis</headline>
        <creationtime>2007-04-15</creationtime>
        <content>Comments on my Master's Thesis.</content>
        <author>Markku Laine</author>
        <comments>
          <comment id="6e75a5517cef0f8d45cfbc280d1acd53">
            <creationtime>2007-09-27</creationtime>
            <content>
              A thorough comparison of related XRX-based
              end-to-end solutions needs to be done.
           </content>
            <author>Markku Laine</author>
          </comment>
          <comment id="83ce80c6cba003108283294928155335">
            <creationtime>2007-10-04</creationtime>
            <content>
              The XFormsDB framework should be properly evaluated
              with real users and case studies at some point.
            </content>
            <author>Emilia Mendes</author>
          </comment>
        </comments>
      </post>
      ...
    </posts>
  </blog>
</root>
```

7.3.8 Metrics

Table 30 shows the component metrics of the Blog Web application. Compared to the PIM Web application, the amount of work needed on CSS and XForms has increased significantly, which can be explained by more stylish layout and a wealth

of dynamic features on the Web site. In addition, the amount of XFormsDB and XQuery & XPath code is much higher due to the use of multiple queries.

Table 30: Blog component metrics

Component	Lines	Elements	Attributes	Rules
XHTML	413	229	268	0
CSS	627	0	0	116
JavaScript	66	0	0	0
XForms	936	703	817	0
Instance Data	156	131	31	0
XFormsDB	123	73	165	0
XQuery and XPath	122	0	0	0
Database Template	10	5	1	0
Comment	305	0	0	0
Empty	113	0	0	0
Miscellaneous	10	0	6	0
Total	**2881**	**1141**	**1288**	**116**

The performance metrics of the Blog Web application are shown in Table 31 and Table 32. The measurements were made with both empty cache and primed cache. The total Web page weight was reduced by 99% and the average Web page load time was reduced by 41-49% on subsequent page views.

Table 31: Blog response size metrics

	Empty Cache		Primed Cache	
	Requests	**Size (kB)**	**Requests**	**Size (kB)**
index.xformsdb				
HTML	1	3.2	1	3.2
JavaScript	3	111.0	2	0.0
CSS	3	8.6	2	0.0
Image	10	363.9	7	0.8
Total	**17**	**486.8**	**12**	**4.1**
login.xformsdb				
HTML	1	2.3	1	2.3
JavaScript	3	111.0	2	0.0
CSS	3	8.6	2	0.0
Image	9	363.8	6	0.8
Total	**16**	**485.7**	**11**	**3.2**
admin/index.xformsdb				
HTML	1	4.6	1	4.6
JavaScript	3	111.0	2	0.0
CSS	3	8.6	2	0.0
Image	12	364.3	9	0.8
Total	**19**	**488.6**	**14**	**5.5**

Table 32: Blog response time metrics

	Empty Cache	Primed Cache
	Time (s)	**Time (s)**
index.xformsdb		
Backend	1.6	0.7
Frontend	0.6	0.5
Total	**2.1**	**1.2**
login.xformsdb		
Backend	1.4	0.5
Frontend	0.6	0.5
Total	**2.0**	**1.0**
admin/index.xformsdb		
Backend	1.7	0.9
Frontend	0.6	0.5
Total	**2.3**	**1.4**

The sizes of the WAR files were 406.1 kB (Lite version) and 17.9 MB (Standalone version).

7.3.9 Analysis

The feasibility of the XFormsDB framework for developing popular real-life Web applications of today was clearly demonstrated by the success of the Blog Web application. Below are listed the salient observations and issues that were raised during the development process of the Blog Web application.

Firstly, the size of an individual Web page should be kept relatively low, since Web pages become progressively more complex to maintain as the number of lines and user interface views on a Web page increases. Secondly, transaction support should be improved. At the moment, transactions are a bit cumbersome because they cannot be grouped—the next request can be reliably submitted only after the previous one has been successfully executed. Finally, the validation proved how easily ready-made functions and queries can be taken in use, which therefore eases Web application development for non-programmers.

Chapter 8

Conclusions

In this Chapter, the research objectives of this Thesis are revisited, after which the main contributions of this Thesis are summarized, the results of the research based on the theoretical and empirical validation techniques are presented, and the main conclusions are drawn. Finally, some possible directions for the future work are discussed.

8.1 Research Objectives Revisited

Before presenting the research results, it is worthwhile to revisit the research objectives of this Thesis as stated in Chapter 4. The main research questions were:

Q1: Is it possible to extend the XForms markup language in such a way that users can build useful, highly interactive multi-user Web applications quickly and easily using purely declarative languages?

Q2: How the extension can be kept simple enough, so that even non-programmers are capable of utilizing it?

Q3: By what means should the feasibility of the extension be validated?

In order to meet the research objectives and to answer the aforementioned research questions, a thorough review of the literature and related work was conducted. Then, requirements for an extension to the XForms markup language were derived and an extension called the XFormsDB markup language was designed. Finally, the feasibility of the designed extension was validated by developing a proof-of-concept implementation, called the XFormsDB framework, and two sample Web applications using the implementation.

8.2 Main Contributions

The main contributions of this Thesis can be summarized as follows:

- Design of a workable declarative language, *the XFormsDB markup language*, for developing multi-user Web applications (including a workable combination of XForms and XQuery), which has been partially designed by the Author
- Implementation of a workable prototype, *the XFormsDB framework*, whose sole implementer the Author has been
- Validation of the designed language and the implemented prototype using two sample Web applications, *Personal Information Management (PIM): Contacts* and *Blog*, which have been entirely developed by the Author

8.3 Results

The XFormsDB markup language proves that the XForms markup language can be naturally extended to include common server-side functionalities, such as data source access as well as authentication and access control. New server-side functionalities can also be easily added to the XFormsDB markup language if necessary, since this requirement has been already taken into account during the design of the language.

For users who are already familiar with XForms, the XFormsDB markup language is most likely relatively easy to learn because the syntax and processing model of the language is similar to XForms as well as it involves only a few additions. The language also takes into account users with varying skills and capabilities, among others, by supporting novice users with the possibility to use ready-made components authored by other users as well as by providing a simple and elegant way for performing synchronized updates. However, for users who are not familiar with the declarative style of programming the learning curve might prove to be somewhat steep mainly due to XForms and other XML technologies it is dependent upon.

The XFormsDB framework in turn verified that the XFormsDB markup language is technically implementable as well as that Web applications which utilize the framework are capable of supporting most common browsers and heterogeneous data sources. The two sample Web applications, which were developed using the XFormsDB framework, clearly demonstrated the feasibility and capabilities of the framework by proving that useful, highly interactive multi-user Web applications can be authored quickly and easily using purely declarative languages. From the end user's point of view, the performance and responsiveness of the applications were very good, especially when considering that neither the framework nor the applications have been optimized.

All in all, the results show that there are no major issues with this new authoring paradigm. The XFormsDB framework proved to be an extremely powerful XRX framework, which allows for the rapid development of entire Web applications using a single document and under a single programming model. Therefore, its use can be highly recommended—especially for non-programmers—if the application scope is within declarative languages.

8.4 Future Work

There are several interesting directions for future work that one could pursue based on the work presented in this Thesis. First of all, the development of XFormsDB Web applications could be further simplified by refining the syntax of the

XFormsDB markup language and by implementing support for a subset of XML Binding Language (XBL) 2.0 [44] on the server side, which would allow the use of highly reusable components containing XFormsDB markup in Web applications. It would be also interesting to study whether a Web-based visual tool for developing XFormsDB Web applications would make the technology accessible to non-technical users as well.

From the end user's point of view, the possibility of being able to use an existing account to sign in to XFormsDB Web applications would be greatly appreciated. Therefore, adding built-in support for OpenID [109] authentication would be very beneficial.

In addition, current transaction support should be improved to support grouping of synchronized updates. Finally, improving the performance of the framework through caching is also something that has to be looked into.

The development of the XFormsDB framework will continue during the year 2010 and it is planned to be released as an open source project later this spring.

Bibliography

[1] Sommerville, I. Software Engineering. 8th ed. Pearson Education Limited, Essex, England: Addison-Wesley, 2007. p. 840. ISBN 0-321-31379-8.

[2] Cardone, R., Soroker, D., and Tiwari, A. Using XForms to Simplify Web Programming. In: Proceedings of the 14th International Conference on World Wide Web. Chiba, Japan. May 10-14, 2005. New York, NY, USA: ACM Press, 2005. p. 215-224. ISBN: 1-59593-046-9.

[3] Yang, F., Gupta, N., Gerner, N., Qi, X., Demers, A., Gehrke, J., and Shanmugasundaram, J. Performance Engineering of Web Applications: A Unified Platform for Data Driven Web Applications with Automatic Client-Server Partitioning. In: Proceedings of the 16th International Conference on World Wide Web. Banff, Alberta, Canada. May 8-12, 2007. New York, NY, USA: ACM Press, 2007. p. 341-350. ISBN: 978-1-59593-654-7.

[4] Google. [Online]. Google Web Toolkit – Google Code. [Cited January 20, 2010]. Available at:
http://code.google.com/webtoolkit/

[5] Schmitz, P. The SMIL 2.0 Timing and Synchronization Model: Using Time in Documents. In: Technical Report MSR-TR-2001-01. Microsoft Research, Microsoft Corporation. Jan 2, 2001. Redmond, WA 98052: One Microsoft Way, 2001.

[6] NextApp, Inc. [Online]. Echo Web Framework | Developer information for creating AJAX web applications using the Echo2 and Echo3 frameworks. [Cited January 20, 2010]. Available at:
http://echo.nextapp.com/site/

[7] Red Hat, Inc. [Online]. hibernate.org – Hibernate. [Cited January 20, 2010]. Available at:
http://www.hibernate.org/

[8] The Apache Software Foundation. [Online]. Welcome to XMLBeans. [Cited January 20, 2010]. Available at:
http://xmlbeans.apache.org/

[9] Honkala, M., Koskimies, O., and Laine, M. Connecting XForms to Databases: An Extension to the XForms Markup Language. In: Position Paper for W3C Workshop on Declarative Models of Distributed Web Applications. Dublin, Ireland. June 5-6, 2007.

[10] Shaw, M. What Makes Good Research in Software Engineering? In: International Journal on Software Tools for Technology Transfer (STTT), 2002. Vol. 4:1. p. 1-7. ISSN 1433-2779.

[11] Malhotra, A. and Maloney, M. [Online]. XML Schema Requirements. W3C Note 15 February 1999. [Cited January 20, 2010]. Available at:
http://www.w3.org/TR/NOTE-xml-schema-req

[12] Heflin, J. [Online]. OWL Web Ontology Language Use Cases and Requirements. W3C Recommendation 10 February 2004. [Cited January 20, 2010]. Available at:
http://www.w3.org/TR/webont-req/

[13] Pixley, T. [Online]. Document Object Model (DOM) Level 2 Events Specification. Version 1.0. W3C Recommendation 13 November, 2000. [Cited January 20, 2010]. Available at:
http://www.w3.org/TR/DOM-Level-2-Events/

[14] McCarron, S., Pemberton, S., and Raman, T.V. [Online]. XML Events. An Events Syntax for XML. W3C Recommendation 14 October 2003. [Cited January 20, 2010]. Available at:
http://www.w3.org/TR/xml-events/

[15] Lawrence, S. and Leach, P. [Online]. User Agent Authentication Forms. W3C Note – 03 Feb 1999. [Cited January 20, 2010]. Available at:
http://www.w3.org/TR/NOTE-authentform

[16] W3C. [Online]. HTTP Authentication – W3C XForms Group Wiki (Public). [Cited January 20, 2010]. Available at:
http://www.w3.org/MarkUp/Forms/wiki/HTTP_Authentication

[17] Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., Yergeau, F., and Cowan, J. [Online]. Extensible Markup Language (XML) 1.1 (Second Edition). W3C Recommendation 16 August 2006, edited in place 29 September 2006. [Cited January 20, 2010]. Available at: http://www.w3.org/TR/xml11/

[18] Dubinko, M. XForms Essentials. O´Reilly & Associates, Inc., Sebastopol, CA, the United States of America: O´Reilly, 2003. p. 215. ISBN: 0-596-00369-2.

[19] Honkala, M. Web User Interaction – A Declarative Approach based on XForms. Thesis (PhD). Telecommunications Software and Multimedia Laboratory, Helsinki University of Technology. Espoo: Otamedia Oy, 2006. 2006. p. 81. ISBN: 951-22-8565-7.

[20] eXforms. [Online]. eXforms – Extend the power of XForms. [Cited January 20, 2010]. Available at: http://www.exforms.org/

[21] Honkala, M. and Vuorimaa, P. Secure Web Forms with Client-Side Signatures. In: Proceedings of the 5th International Conference on Web Engineering, ICWE 2005. Sydney, Australia. July 27-29, 2005. Berlin / Heidelberg, Germany: SpringerLink, 2005. p. 340-351. ISBN: 978-3-540-27996-9.

[22] Bourret, R. [Online]. rpbourret.com – XML Database Products: Middleware. [Cited January 20, 2010]. Available at: http://www.rpbourret.com/xml/ProdsMiddleware.htm

[23] Honkala, M. and Vuorimaa, P. XForms in X-Smiles. In: Journal of World Wide Web, 2001. Vol. 4:3. p. 151-166. ISSN: 1386-145X.

[24] X-Smiles.org *et al.* [Online]. X-Smiles.org. [Cited January 20, 2010]. Available at: http://www.xsmiles.org/

[25] Michel, T. [Online]. XForms Implementation Report. [Cited January 20, 2010]. Available at: http://www.w3.org/MarkUp/Forms/Test/ImplementationReport.html

[26] mozilla.org. [Online]. Mozilla XForms Project. [Cited January 20, 2010]. Available at: http://www.mozilla.org/projects/xforms/

[27] x-port.net Ltd. [Online]. XForms processor from formsPlayer | standards. innovation. [Cited January 20, 2010]. Available at: http://www.formsplayer.com/

[28] côdeazur brazil. [Online]. DENG – The Modular XML Browser Engine – CSS 3 plus XHTML, SVG, XForms, XFrames, RSS and more. [Cited January 20, 2010]. Available at: http://deng.com.br/

[29] Progeny Systems Corporation. [Online]. FormFaces.com. [Cited January 20, 2010]. Available at: http://www.formfaces.com/

[30] McCreary, D. [Online]. Dr. Data Dictionary: Introducing the XRX Architecture: XForms/REST/XQuery. December 14, 2007. [Cited January 20, 2010]. Available at: http://datadictionary.blogspot.com/2007/12/introducing-xrx-architecture.html

[31] Chiba. [Online]. Chiba Home. [Cited January 20, 2010]. Available at: http://www.chiba-project.org/chiba/

[32] AJAXForms S.L. [Online]. AJAXForms – Home. [Cited January 20, 2010]. Available at: http://ajaxforms.sourceforge.net/

[33] Pemberton, S. and Boyer, J. [Online]. The Forms Working Group. [Cited January 20, 2010]. Available at: http://www.w3.org/MarkUp/Forms/

[34] Krasner, G.E. and pope, S.T. A Cookbook for using the Model-View-Controller User Interface Paradigm in Smalltalk-80. In: Journal of Object-Oriented Programming, August/September 1988. Vol. 1:3. p. 26-49. ISSN: 0896-8438.

[35] Dubinko, M., Klotz, Jr., L.L., Merrick, R., and Raman, T.V. [Online]. XForms 1.0. W3C Recommendation 14 October 2003. [Cited January 20, 2010]. Available at: http://www.w3.org/TR/2003/REC-xforms-20031014/

[36] Gray, J. and Reuter, A. Transaction processing: concepts and techniques. Morgan Kaufmann Publishers, Inc., San Francisco, CA, the United States of America: Morgan Kaufmann Publishers, Inc., 1993. p. 1070. ISBN: 1-55860-190-2.

[37] Boyer, J., Bray, T., and Gordon, M. [Online]. Extensible Forms Description Language (XFDL) 4.0. W3C Note, September 2, 1998. [Cited January 20, 2010]. Available at: http://www.w3.org/TR/NOTE-XFDL

[38] McKenzie, G.F., Devitt, R., McDougall, R., Miller, A., Neilson, A., and Tardif, M. [Online]. XFA-Template. Version 1.0. [Cited January 20, 2010]. Available at: http://www.w3.org/1999/05/XFA/xfa-template

[39] McKenzie, G.F., Tardif, M., Devitt, R., McDougall, R., Miller, A., and Neilson, A. [Online]. XFA-FormCalc. Version 1.0. [Cited January 20, 2010]. Available at: http://www.w3.org/1999/05/XFA/xfa-formcalc-19990614

[40] Boyer, J.M. [Online]. XForms 1.1. W3C Recommendation 20 October 2009. [Cited January 20, 2010]. Available at: http://www.w3.org/TR/xforms11/

[41] Garrett, J.J., Adaptive Path. [Online]. adaptive path >> ajax: a new approach to web applications. February 18, 2005. [Cited January 20, 2010]. Available at: http://www.adaptivepath.com/ideas/essays/archives/000385.php

[42] van Kesteren, A. [Online]. XMLHttpRequest. W3C Working Draft 19 November 2009. [Cited January 20, 2010]. Available at: http://www.w3.org/TR/XMLHttpRequest/

[43] Prototype Core Team. [Online]. Prototype JavaScript framework: Easy Ajax and DOM manipulation for dynamic web applications. [Cited January 20, 2010]. Available at: http://www.prototypejs.org/

[44] Hickson, I. [Online]. XML Binding Language (XBL) 2.0. W3C Candidate Recommendation 16 March 2007. [Cited January 20, 2010]. Available at: http://www.w3.org/TR/xbl/

[45] Connolly, D. [Online]. HyperText Mark-up Language. [Cited January 20, 2010]. Available at: http://www.w3.org/History/19921103-hypertext/hypertext/WWW/MarkUp/MarkUp.html

[46] Raggett, D., Lam, J., Alexander, I., and Kmiec, M. Raggett on HTML 4. 2nd ed. Reading, Mass.: Addison-Wesley, 1998. p. 437. ISBN: 0-201-17805-2.

[47] Raggett, D., le Hors, A., and Jacobs, I. [Online]. HTML 4.01 Specification. W3C Recommendation 24 December 1999. [Cited January 20, 2010]. Available at:
http://www.w3.org/TR/html401/

[48] W3C. [Online]. W3C XHTML2 Working Group Home Page. [Cited January 20, 2010]. Available at:
http://www.w3.org/MarkUp/

[49] W3C. [Online]. XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition). A Reformulation of HTML 4 in XML 1.0. W3C Recommendation 26 January 2000, revised 1 August 2002. [Cited January 20, 2010]. Available at:
http://www.w3.org/TR/xhtml1/

[50] Baker, M., Ishikawa, M., Matsui, S., Stark, P., Wugofski, T., and Yamakami, T. [Online]. XHTML™ Basic 1.1. W3C Recommendation 29 July 2008. [Cited January 20, 2010]. Available at:
http://www.w3.org/TR/xhtml-basic/

[51] Altheim, M. and McCarron, S. [Online]. XHTML™ 1.1 – Module-based XHTML. W3C Recommendation 31 May 2001. [Cited January 20, 2010]. Available at:
http://www.w3.org/TR/xhtml11/

[52] Axelsson, J., Birbeck, M., Dubinko, M., Epperson, B., Ishikawa, M., McCarron, S., Navarro, A., and Pemberton, S. [Online]. XHTML™ 2.0. W3C Working Draft 26 July 2006. [Cited January 20, 2010]. Available at:
http://www.w3.org/TR/xhtml2/

[53] Bos, B., Çelik, T., Hickson, I., and Wium Lie, H. [Online]. Cascading Style Sheets, Level 2 Revision 1 (CSS 2.1) Specification. W3C Candidate Recommendation 08 September 2009. [Cited January 20, 2010]. Available at:
http://www.w3.org/TR/CSS2/

[54] Burke, E.M. Java™ and XSLT. O´Reilly & Associates, Inc., Sebastopol, CA, the United States of America: O´Reilly, 2001. p. 510. ISBN: 0-596-00143-6.

[55] Kepser, S. A Simple Proof for the Turing-Completeness of XSLT and XQuery. In: Proceedings of Extreme Markup Languages 2004. Montréal, Québec, Canada. 2004.

[56] Kay, M. [Online]. XSL Transformations (XSLT) Version 2.0. W3C Recommendation 23 January 2007. [Cited January 20, 2010]. Available at: http://www.w3.org/TR/xslt20/

[57] W3C [Online]. The Extensible Stylesheet Language Family (XSL). [Cited January 20, 2010]. Available at: http://www.w3.org/Style/XSL/

[58] Kay, M. [Online]. The SAXON XSLT and XQuery Processor. [Cited January 20, 2010]. Available at: http://saxon.sourceforge.net/

[59] Berners-Lee, T. and Mendelsohn, N. [Online]. The Rule of Least Power. TAG Finding 23 February 2006. [Cited January 20, 2010]. Available at: http://www.w3.org/2001/tag/doc/leastPower.html

[60] International Organization for Standardization. [Online]. ISO 8879:1986 – Information processing -- Text and office systems -- Standard Generalized Markup Language (SGML). [Cited January 20, 2010]. Available at: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=16387

[61] Fallside, D.C. and Walmsley, P. [Online]. XML Schema Part 0: Primer Second Edition. W3C Recommendation 28 October 2004. [Cited January 20, 2010]. Available at: http://www.w3.org/TR/xmlschema-0/

[62] W3C. [Online]. W3C Document Object Model. [Cited January 20, 2010]. Available at: http://www.w3.org/DOM/

[63] The SAX Project. [Online]. SAX. [Cited January 20, 2010]. Available at: http://www.saxproject.org/

[64] W3C. [Online]. Web Services @ W3C. [Cited January 20, 2010]. Available at: http://www.w3.org/2002/ws/

[65] W3C. [Online]. HTTP – Hypertext Transfer Protocol Overview. [Cited January 20, 2010]. Available at: http://www.w3.org/Protocols/

[66] Ecma international. [Online]. ECMAScript Language Specification. Standard ECMA-262. 5th Edition / December 2009. [Cited January 20, 2010]. Available at: http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf

[67] The Apache Software Foundation. [Online]. Apache Ant – Welcome. [Cited January 20, 2010]. Available at: http://ant.apache.org/

[68] The Apache Software Foundation. [Online]. Apache Tomcat – Welcome! [Cited January 20, 2010]. Available at: http://tomcat.apache.org/

[69] The eXist Project. [Online]. eXist-db Open Source Native XML Database. [Cited January 20, 2010]. Available at: http://www.exist-db.org/

[70] Lindholm, T. [Online]. BerliOS Developer: Project Summary – 3DM XML diff and merge tool. [Cited January 20, 2010]. Available at: http://developer.berlios.de/projects/tdm/

[71] Sun Microsystems, Inc. [Online]. Java SE Technologies at a Glance. [Cited January 20, 2010]. Available at: http://java.sun.com/javase/technologies/index.jsp

[72] Orbeon, Inc. [Online]. Orbeon Forms – Web Forms for the Enterprise, Done the Right Way. [Cited January 20, 2010]. Available at: http://www.orbeon.com/

[73] Orbeon, Inc. [Online]. XForms – Using the Orbeon Forms XForms Engine with Java Applications (forms). [Cited January 20, 2010]. Available at: http://wiki.orbeon.com/forms/doc/developer-guide/xforms-with-java-applications

[74] Bruchez, E., Orbeon, Inc. [Online]. XTech 2006: XForms: an Alternative to Ajax? In: XTech 2006: "Building Web 2.0". Amsterdam, The Netherlands. May 16-19, 2006. [Cited January 20, 2010]. Available at: http://xtech06.usefulinc.com/schedule/paper/133

[75] The XML:DB Initiative. [Online]. XML:DB Initiative: Enterprise Technologies for XML Databases. [Cited January 20, 2010]. Available at: http://xmldb-org.sourceforge.net/

[76] Winer, D. [Online]. XML-RPC Specification. Tue, Jun 15, 1999. [Cited January 20, 2010]. Available at:
http://www.xmlrpc.com/spec

[77] The eXist Project. [Online]. Database Deployment. [Cited January 20, 2010]. Available at:
http://www.exist-db.org/deployment.html

[78] Fielding, R.T. Architectural Styles and the Design of Network-based Software Architectures. Thesis (PhD). Information and Computer Science, University of California, Irwine, 2000.

[79] Datypic. [Online]. FunctX XQuery Functions: Hundreds of useful examples. [Cited January 20, 2010]. Available at:
http://www.xqueryfunctions.com/

[80] Ant-Contrib Project. [Online]. Ant-Contrib Tasks. [Cited January 20, 2010]. Available at:
http://ant-contrib.sourceforge.net/

[81] Eclipse Foundation, Inc. [Online]. Eclipse.org home. [Cited January 20, 2010]. Available at:
http://www.eclipse.org/

[82] Tigris.org. [Online]. subclipse.tigris.org. [Cited January 20, 2010]. Available at:
http://subclipse.tigris.org/

[83] Sun Microsystems, Inc. [Online]. Scaling The N-Tier Architecture. White Paper. Sun Microsystems, Inc. September 2000. [Cited January 20, 2010]. Available at:
http://www.sun.com/software/whitepapers/wp-ntier/wp-ntier.pdf

[84] Sun Microsystems, Inc. [Online]. Questions and Answers – Session state in the client tier. [Cited January 20, 2010]. Available at:
http://java.sun.com/blueprints/qanda/client_tier/session_state.html

[85] Lindholm, T. A Three-way Merge for XML Documents. In: Proceedings of the 2004 ACM Symposium on Document Engineering, DocEng'04. Milwaukee, Wisconsin, USA. October 28-30, 2004. New York, NY, USA: ACM, 2004. p. 1-10. ISBN: 1-58113-938-1.

[86] Bourret, R. [Online]. rpbourret.com – XML and Databases. [Cited January 20, 2010]. Available at: http://www.rpbourret.com/xml/XMLAndDatabases.htm

[87] The XML:DB Initiative. [Online]. Frequently Asked Questions About XML:DB. [Cited January 20, 2010]. Available at: http://xmldb-org.sourceforge.net/faqs.html

[88] Software AG. [Online]. Tamino – The XML Database. [Cited January 20, 2010]. Available at: http://www.softwareag.com/Corporate/products/wm/tamino/default.asp

[89] Microsoft Corporation. [Online]. SQL Server 2008 Overview, data platform, store data | Microsoft. [Cited January 20, 2010]. Available at: http://www.microsoft.com/sqlserver/2008/en/us/default.aspx

[90] IBM Corporation. [Online]. IBM – DB2 – Data server – database software – database management – open source. [Cited January 20, 2010]. Available at: http://www-01.ibm.com/software/data/db2/

[91] Oracle Corporation. [Online]. Database 11g | Oracle Database 11g | Oracle. [Cited January 20, 2010]. Available at: http://www.oracle.com/us/products/database/index.htm

[92] Bourret, R. [Online]. rpbourret.com – XML Database Products: XML-Enabled Databases. [Cited January 20, 2010]. Available at: http://www.rpbourret.com/xml/ProdsXMLEnabled.htm

[93] Codd, E.F. A Relational Model of Data for Large Shared Data Banks. In: Communications of the ACM. New York, NY, USA: ACM, 1970. Vol. 13:6. p. 377-387. ISSN: 0001-0782. [doi: http://doi.acm.org/10.1145/362384.362685].

[94] Digital Equipment Corporation. [Online]. Information Technology – Database Language SQL (Proposed revised text of DIS 9075), July 1992. [Cited January 20, 2010]. Available at: http://www.contrib.andrew.cmu.edu/~shadow/sql/sql1992.txt

[95] American National Standards Institute. [Online]. American National Standards Institute – ANSI. [Cited January 20, 2010]. Available at: http://www.ansi.org/

[96] International Organization for Standardization. [Online]. ISO – International Organization for Standardization. [Cited January 20, 2010]. Available at: http://www.iso.org/iso/home.htm

[97] Leavitt, N. [Online]. Whatever Happened to Object-Oriented Databases? In: Computer. Los Alamitos, CA, USA: IEEE Computer Society Press, 2000. Vol. 33:8. p. 16-19. ISSN: 0018-9162. [Cited January 20, 2010]. Available at: http://www.leavcom.com/pdf/DBpdf.pdf

[98] ISO. [Online]. Information Technology – Database Languages SQL – Part 14: XML-Related Specifications (SQL/XML), July 2005. [Cited January 20, 2010]. Available at: http://www.sqlx.org/SQL-XML-documents/5FCD-14-XML-2004-07.pdf

[99] SQLX.org. [Online]. Definition of SQLX.org and SQL/XML. [Cited January 20, 2010]. Available at: http://www.sqlx.org/SQL-XML-definition/SQL-XML-definition.html

[100] Sun Microsystems, Inc. [Online]. Java SE Technologies – Database. [Cited January 20, 2010]. Available at: http://java.sun.com/javase/technologies/database/

[101] Progress Software Corporation. [Online]. SQL/XML Tutorial. [Cited January 20, 2010]. Available at: http://www.stylusstudio.com/sqlxml_tutorial.html

[102] Marsh, J., Orchard, D., and Veillard, D. [Online]. XML Inclusions (XInclude) Version 1.0 (Second Edition). W3C Recommendation 15 November 2006. [Cited January 20, 2010]. Available at: http://www.w3.org/TR/xinclude/

[103] XK72 Ltd. [Online]. Charles Web Debugging Proxy • HTTP Monitor / HTTP Proxy / HTTPS & SSL Proxy / Reverse Proxy. [Cited January 20, 2010]. Available at: http://www.charlesproxy.com/

[104] Souders, S. [Online]. Episodes. [Cited January 20, 2010]. Available at: http://stevesouders.com/episodes/

[105] Souders, S. [Online]. Episodes: a Framework for Measuring Web Page Load Times. White Paper. Google. July 2008. [Cited January 20, 2010]. Available at: http://stevesouders.com/episodes/paper.php

[106] GNU Project. [Online]. The gzip home page. [Cited January 20, 2010]. Available at:
http://www.gzip.org/

[107] Hickson, I. and Hyatt, D. [Online]. HTML 5. A vocabulary and associated APIs for HTML and XHTML. W3C Working Draft 25 August 2009. [Cited January 20, 2010]. Available at:
http://www.w3.org/TR/html5/

[108] Florescu, D. and Kossmann, D. A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database. In: Technical Report No 3680, INRIA. May, 1999. ISSN: 0249-6399.

[109] OpenID Foundation. [Online]. OpenID Foundation website. [Cited January 20, 2010]. Available at:
http://openid.net/

[110] Berglund, A., Boag, S., Chamberlin, D., Fernández, M., Kay, M., Robie, J., and Siméon, J. [Online]. XML Path Language (XPath) 2.0. W3C Recommendation 23 January 2007. [Cited January 20, 2010]. Available at:
http://www.w3.org/TR/xpath20/

[111] Chamberlin, D., Robie, J., and Florescu, D. Quilt: An XML Query Language for Heterogeneous Data Sources. In: Proceedings of WebDB 2000 Conference, in Lecture Notes in Computer Science, Springer-Verlag, 2000.

[112] Robie, J., Lapp, J., and Schach, D. [Online]. XML Query Language (XQL). [Cited January 20, 2010]. Available at:
http://www.w3.org/TandS/QL/QL98/pp/xql.html

[113] Deutsch, A., Fernandez, M., Florescu, D., Levy, A., and Suciu, D. [Online]. XML-QL_ A Query Language for XML. Submission to the World Wide Web Consortium 19-August-1998. [Cited January 20, 2010]. Available at:
http://www.w3.org/TR/NOTE-xml-ql/

[114] Cattell, R.G.G., Barry, D.K., Berler, M., Eastman, J., Jordan, D., Russell, C., Schadow, O, Stanienda, T., and Velez, F. The Object Data Management Standard: ODMG 3.0. Morgan Kaufmann Publishers, Inc., San Francisco, CA, the United States of America: Morgan Kaufmann Publishers, Inc., 2000. p. 300. ISBN: 1-55860-647-5.

[115] Fernández, M., Malhotra, A., Marsh, J., Nagy, M., and Walsh, N. [Online]. XQuery 1.0 and XPath 2.0 Data Model (XDM). W3C Recommendation 23 January 2007. [Cited January 20, 2010]. Available at: http://www.w3.org/TR/xpath-datamodel/

[116] Boag, S., Chamberlin, D., Fernández, M., Florescu, D., Robie, J., and Siméon, J. [Online]. XQuery 1.0: An XML Query Language. W3C Recommendation 23 January 2007. [Cited January 20, 2010]. Available at: http://www.w3.org/TR/xquery/

[117] Boag, S., Kay, M., Tong, J., Walsh, N., and Zongaro, H. [Online]. XSLT 2.0 and XQuery 1.0 Serialization. W3C Recommendation 23 January 2007. [Cited January 20, 2010]. Available at: http://www.w3.org/TR/xslt-xquery-serialization/

[118] Malhotra, A., Melton, J., and Walsh, N. [Online]. XQuery 1.0 and XPath 2.0 Functions and Operators. W3C Recommendation 23 January 2007. [Cited January 20, 2010]. Available at: http://www.w3.org/TR/xpath-functions/

[119] Melton, J. and Muralidhar, S. [Online]. XML Syntax for XQuery 1.0 (XQueryX). W3C Recommendation 23 January 2007. [Cited January 20, 2010]. Available at: http://www.w3.org/TR/xqueryx/

[120] Chamberlin, D., Dyck, M., Florescu, D., Melton, J., Robie, J., and Siméon, J. [Online]. XQuery Update Facility 1.0. W3C Candidate Recommendation 09 June 2009. [Cited January 20, 2010]. Available at: http://www.w3.org/TR/xquery-update-10/

[121] Amer-Yahia, S., Botev, C., Buxton, S., Case, P., Doerre, J., Dyck, M., Holstege, M., Melton, J., Rys, M., and Shanmugasundaram, J. [Online]. XQuery and XPath Full Text 1.0. W3C Candidate Recommendation 09 July 2009. [Cited January 20, 2010]. Available at: http://www.w3.org/TR/xpath-full-text-10/

[122] Chamberlin, D., Engovatov, D., Florescu, D., Ghelli, G., Melton, J., Siméon, J., and Snelson, J. [Online]. XQuery Scripting Extension 1.0. W3C Working Draft 3 December 2008. [Cited January 20, 2010]. Available at: http://www.w3.org/TR/xquery-sx-10/

[123] Melton, J. and Van Cappellen, M. [Online]. XQuery API for Java™ (XQJ) 1.0 Specification (Final Release). March, 2009. [Cited January 20, 2010]. Available at: http://jcp.org/aboutJava/communityprocess/final/jsr225/index.html

[124] Bray, T., Hollander, D., Layman, A., and Tobin, R. [Online]. Namespaces in XML 1.1 (Second Edition). W3C Recommendation 16 August 2006. [Cited January 20, 2010]. Available at: http://www.w3.org/TR/xml-names11/

[125] Progress Software Corporation. [Online]. DataDirect XQuery Product Overview. [Cited January 20, 2010]. Available at: http://www.xquery.com/xquery/

[126] W3C. [Online]. World Wide Web Consortium (W3C). [Cited January 20, 2010]. Available at: http://www.w3.org/

[127] Sun Microsystems, Inc. [Online]. Core J2EE Patterns – Front Controller. [Cited January 20, 2010]. Available at: http://java.sun.com/blueprints/corej2eepatterns/Patterns/FrontController.html

[128] Jazayeri, M. Some Trends in Web Application Development. In: Future of Software Engineering (FOSE'07). Washington, DC, USA: IEEE Computer Society, 2007. p. 199-213. ISBN: 0-7695-2829-5.

[129] Duhl, J. [Online]. Rich Internet Applications. White Paper. November 2003. [Cited January 20, 2010]. Available at: http://www.adobe.com/platform/whitepapers/idc_impact_of_rias.pdf

[130] Preciado, J.C., Linaje, M., Sánchez, F., and Comai, S. Necessity of methodologies to model Rich Internet Applications. In: Proceedings of the Seventh IEEE International Symposium on Web Site Evolution (WSE'05). IEEE Computer Society, 2005. p. 7-13. DOI: 10.1109/WSE.2005.10.

[131] Mikkonen, T. and Taivalsaari, A. [Online]. Web Applications – Spaghetti Code for the 21st Century. Technical Report. Sun Microsystems, Inc. June 2007. [Cited January 20, 2010]. Available at: http://research.sun.com/techrep/2007/smli_tr-2007-166.pdf

[132] Kuusteri, J. and Mikkonen, T. Partitioning Web Applications Between the Server and the Client. In: Proceedings of the 2009 ACM symposium on Applied Computing (SAC'09). Honolulu, Hawaii, USA. March 8-12, 2009. p. 647-652. ISBN: 978-1-60558-166-8.

[133] Lloyd, J.W. [Online]. Declarative Programming in Escher. Technical Report. University of Bristol, Bristol, UK. June 1995. [Cited January 20, 2010]. Available at: http://www.cs.bris.ac.uk/Publications/Papers/1000073.pdf

[134] Birbeck, M. [Online]. Application of XML Access: XForms and XQuery, via REST. Presentation. [Cited January 20, 2010]. Available at: http://www.w3c.rl.ac.uk/pastevents/XML_Access_Languages/Mark/xforms-xquery.html

[135] Sun Microsystems, Inc. [Online]. JavaServer Pages Technology. [Cited January 20, 2010]. Available at: http://java.sun.com/products/jsp/

Appendix A

Syntax Definitions and Usage Examples of the XFormsDB Markup Language

For the simplicity of the underlying examples, all XML Namespace declarations have been left out.

Listing 15: Definition of the *xformsdb:instance* element

```
<!-- XFormsDB request instance -->
<xformsdb:instance id="xformsdb-request-instance">
  <!-- Actions defined by the form author -->
</xformsdb:instance>
```

Listing 16: Example of use of the *state* request for storing a Web application's state information in an XFormsDB implementation

```
<!-- State instance -->
<xforms:instance id="state-instance">
  <ui xmlns="">
    <language>en</language>
    <text-size>medium</text-size>
  </ui>
</xforms:instance>

<!-- State request instance -->
<xformsdb:instance id="state-request-instance">
  <xformsdb:state />
</xformsdb:instance>

<!-- Set state request submission -->
<xformsdb:submission id="set-state-request-submission"
  replace="instance" instance="state-instance"
  requestinstance="state-request-instance" statetype="set"
  attachmentinstance="state-instance">
  <!-- Actions defined by the form author -->
  <xforms:action ev:event="xformsdb-request-error">
    <!-- Show an error message -->
    <xforms:toggle case="error-set-state-case" />
  </xforms:action>
  <xforms:action ev:event="xforms-submit-done">
    <!-- Show a success message -->
    <xforms:toggle case="success-set-state-case" />
  </xforms:action>
</xformsdb:submission>
```

Listing 17: Example of use of the *state* request for retrieving a Web application's state information from an XFormsDB implementation

```
<!-- State instance -->
<xforms:instance id="state-instance">
  <dummy xmlns="" />
</xforms:instance>

<!-- State request instance -->
<xformsdb:instance id="state-request-instance">
  <xformsdb:state />
</xformsdb:instance>

<!-- Get state request submission -->
<xformsdb:submission id="get-state-request-submission"
  replace="instance" instance="state-instance"
  requestinstance="state-request-instance" statetype="get">
  <!-- Actions defined by the form author -->
  <xforms:action ev:event="xformsdb-request-error">
    <!-- Show an error message -->
    <xforms:toggle case="error-get-state-case" />
  </xforms:action>
  <xforms:action ev:event="xforms-submit-done">
    <!-- Show a success message -->
    <xforms:toggle case="success-get-state-case" />
  </xforms:action>
</xformsdb:submission>
```

Listing 18: Example of an XML response indicating a successful completion of the *state* request

```
<?xml version="1.0" encoding="UTF-8"?>
<ui>
  <language>en</language>
  <text-size>medium</text-size>
</ui>
```

Listing 19: Example of use of the *login* request taken from a */login.xformsdb* Web page

```
<!-- User instance -->
<xforms:instance id="user-instance">
  <dummy xmlns="" />
</xforms:instance>

<!-- Login request instance -->
<xformsdb:instance id="login-request-instance">
  <xformsdb:login datasrc="realm-data-source"
    doc="xformsdb_users.xml">
    <xformsdb:var name="username" />
    <xformsdb:var name="password" />
  </xformsdb:login>
</xformsdb:instance>

<!-- Login request submission -->
<xformsdb:submission id="login-request-submission"
  replace="instance" instance="user-instance"
  requestinstance="login-request-instance">
  <!-- Actions defined by the form author -->
  <xforms:action ev:event="xformsdb-request-error">
    <!-- Show an error message -->
    <xforms:toggle case="error-login-case" />
  </xforms:action>
  <xforms:action ev:event="xforms-submit-done">
    <!-- Show an incorrect message -->
    <xforms:toggle if="not( exists(
      instance( 'user-instance' )/@username ) )"
      case="incorrect-login-case" />
    <!-- Show a success message -->
    <xforms:toggle if="exists(
      instance( 'user-instance' )/@username )"
      case="success-login-case" />
    <!-- Redirect the user to /admin/index.xformsdb
         upon a successful login -->
    <xforms:load if="exists(
      instance( 'user-instance' )/@username )"
      resource="/admin/index.xformsdb" />
  </xforms:action>
</xformsdb:submission>
```

Listing 20: Example of an XML response indicating a successful completion of the *login* request

```
<?xml version="1.0" encoding="UTF-8"?>
<xformsdb:user username="administrator" roles="user admin" />
```

Listing 21: Example of an XML response of the *login* request indicating an incorrect username and password combination

```
<?xml version="1.0" encoding="UTF-8"?>
<xformsdb:user />
```

Listing 22: Example of use of the *logout* request

```
<!-- User instance -->
<xforms:instance id="user-instance">
  <dummy xmlns="" />
</xforms:instance>

<!-- Logout request instance -->
<xformsdb:instance id="logout-request-instance">
  <xformsdb:logout />
</xformsdb:instance>

<!-- Logout request submission -->
<xformsdb:submission id="logout-request-submission"
  replace="instance" instance="user-instance"
  requestinstance="logout-request-instance">
  <!-- Actions defined by the form author -->
  <xforms:action ev:event="xformsdb-request-error">
    <!-- Show an error message -->
    <xforms:toggle case="error-logout-case" />
  </xforms:action>
  <xforms:action ev:event="xforms-submit-done">
    <!-- Show a success message -->
    <xforms:toggle case="success-logout-case" />
    <!-- Redirect the user to the home page
         upon a successful logout -->
    <xforms:load resource="/index.xformsdb" />
  </xforms:action>
</xformsdb:submission>
```

Listing 23: Example of an XML response indicating a successful completion of the *logout* request

```
<?xml version="1.0" encoding="UTF-8"?>
<xformsdb:user />
```

Listing 24: Example of use of the *user* request

```
<!-- User instance -->
<xforms:instance id="user-instance">
  <dummy xmlns="" />
</xforms:instance>

<!-- User request instance -->
<xformsdb:instance id="user-request-instance">
  <xformsdb:user />
</xformsdb:instance>

<!-- User request submission -->
<xformsdb:submission id="user-request-submission"
  replace="instance" instance="user-instance"
  requestinstance="user-request-instance">
  <!-- Actions defined by the form author -->
  <xforms:action ev:event="xformsdb-request-error">
    <!-- Show an error message -->
    <xforms:toggle case="error-user-case" />
  </xforms:action>
  <xforms:action ev:event="xforms-submit-done">
    <!-- Show an empty message -->
    <xforms:toggle if="not( exists(
      instance( 'user-instance' )/@username ) )"
      case="empty-user-case" />
    <!-- Show a success message -->
    <xforms:toggle if="exists(
      instance( 'user-instance' )/@username )"
      case="success-user-case" />
  </xforms:action>
</xformsdb:submission>
```

Listing 25: Example of an XML response of the *user* request containing information about the currently logged-in user

```
<?xml version="1.0" encoding="UTF-8"?>
<xformsdb:user username="administrator" roles="user admin" />
```

Listing 26: Example of an XML response of the *user* request indicating that an XFormsDB implementation does not hold a logged-in user in the session

```
<?xml version="1.0" encoding="UTF-8"?>
<xformsdb:user />
```

Listing 27: Example of use of the *query* request for retrieving data from a data source with XQuery

```
<!-- Course instance -->
<xforms:instance id="course-instance">
  <dummy xmlns="" />
</xforms:instance>

<!-- Select course request instance -->
<xformsdb:instance id="select-course-request-instance">
  <xformsdb:query datasrc="courses-data-source" doc="courses.xml">
    <xformsdb:expression>
      <![CDATA[
        xquery version "1.0" encoding "UTF-8";

       (: Declare namespaces :)
       declare namespace studies = "http://www.tkk.fi/2009/studies";
       (: Declare external variables :)
       declare variable $code as xs:string external;

       (: Select course :)
       for $course in /root/studies:courses/studies:course
         where
           $course/studies:code = $code
         return
           $course
      ]]>
    </xformsdb:expression>
    <xformsdb:var name="code" />
  </xformsdb:query>
</xformsdb:instance>

<!-- Select course request submission -->
<xformsdb:submission id="select-course-request-submission"
  replace="instance" instance="course-instance"
  requestinstance="select-course-request-instance"
  expressiontype="select">
  <!-- Actions defined by the form author -->
  <xforms:action ev:event="xformsdb-request-error">
    <!-- Show an error message -->
    <xforms:toggle case="error-select-course-case" />
  </xforms:action>
  <xforms:action ev:event="xforms-submit-done">
```

```
    <!-- Show a success message -->
    <xforms:toggle case="success-select-course-case" />
  </xforms:action>
</xformsdb:submission>
```

Listing 28: Example of an XML response of the *query* request containing the course information

```
<?xml version="1.0" encoding="UTF-8"?>
<studies:course>
  <studies:code>T-111.5360</studies:code>
  <studies:name>WWW Applications</studies:name>
  <studies:credits>4</studies:credits>
  <studies:overview>
    The focus of the course is on new Web technologies.
  </studies:overview>
</studies:course>
```

Listing 29: Example of use of the *query* request for updating data in a data source with XPath

```
<!-- Course instance -->
<xforms:instance id="course-instance">
  <dummy xmlns="" />
</xforms:instance>

<!-- Select and update course request instance -->
<xformsdb:instance id="select-and-update-course-request-instance">
  <xformsdb:query datasrc="courses-data-source" doc="courses.xml">
    <xformsdb:expression>
      /root/studies:courses/studies:course[ studies:code = $code ]
    </xformsdb:expression>
    <xformsdb:xmlns prefix="studies"
       uri="http://www.tkk.fi/2009/studies" />
    <xformsdb:var name="code" />
  </xformsdb:query>
</xformsdb:instance>

<!-- Select course request submission -->
<xformsdb:submission id="select-course-request-submission"
  replace="instance" instance="course-instance"
  requestinstance="select-and-update-course-request-instance"
  expressiontype="select">
  <!-- Actions defined by the form author -->
  <xforms:action ev:event="xformsdb-request-error">
    <!-- Show an error message -->
    <xforms:toggle case="error-select-course-case" />
  </xforms:action>
  <xforms:action ev:event="xforms-submit-done">
    <!-- Show a success message -->
    <xforms:toggle case="success-select-course-case" />
  </xforms:action>
</xformsdb:submission>

<!-- Update course request submission -->
<xformsdb:submission id="update-course-request-submission"
  replace="instance" instance="course-instance"
  requestinstance="select-and-update-course-request-instance"
  expressiontype="update"
  attachmentinstance="course-instance">
  <!-- Actions defined by the form author -->
```

```
  <xforms:action ev:event="xformsdb-request-error">
    <!-- Show an error message -->
    <xforms:toggle case="error-update-course-case" />
  </xforms:action>
  <xforms:action ev:event="xforms-submit-done">
    <!-- Show a success message -->
    <xforms:toggle case="success-update-course-case" />
  </xforms:action>
</xformsdb:submission>
```

Listing 30: Example of an XML response of the *query* request containing the updated course information

```
<?xml version="1.0" encoding="UTF-8"?>
<studies:course>
  <studies:code>T-111.5360</studies:code>
  <studies:name>WWW Applications</studies:name>
  <studies:credits>4</studies:credits>
  <studies:overview>
    The focus of the course is on both new and upcoming Web
    technologies.
  </studies:overview>
</studies:course>
```

Listing 31: Example of use of the *file* request for retrieving the metadata about all public files associated with a Web application

```
<!-- Files instance -->
<xforms:instance id="files-instance">
  <dummy xmlns="" />
</xforms:instance>

<!-- Select/Update/Insert/Delete files request instance -->
<xformsdb:instance id="suid-files-request-instance">
  <xformsdb:file>
    <!-- All public files -->
    <xformsdb:var name="roles" />
  </xformsdb:file>
</xformsdb:instance>

<!-- Select files request submission -->
<xformsdb:submission id="select-files-request-submission"
  replace="instance" instance="files-instance"
  requestinstance="suid-files-request-instance" filetype="select">
  <!-- Actions defined by the form author -->
  <xforms:action ev:event="xformsdb-request-error">
    <!-- Show an error message -->
    <xforms:toggle case="error-select-files-case" />
  </xforms:action>
  <xforms:action ev:event="xforms-submit-done">
    <!-- Show an empty message -->
    <xforms:toggle if="count(
      instance( 'files-instance' )/xformsdb:file ) = 0"
      case="empty-select-files-case" />
    <!-- Show a success message -->
    <xforms:toggle if="count(
      instance( 'files-instance' )/xformsdb:file ) &gt; 0"
      case="success-select-files-case" />
  </xforms:action>
</xformsdb:submission>
```

Listing 32: Example of an XML response of the *file* request containing the metadata about all public files associated with a Web application

```
<?xml version="1.0" encoding="UTF-8"?>
<xformsdb:files>
  <xformsdb:file displayname="API for XFormsDB 1.0" roles=""
    filename="api_for_xformsdb_1-0.pdf" mediatype="application/pdf"
    filesize="144941" comment="Editor's Draft"
    creator="Markku Laine" created="2009-01-27T17:04:55.310+02:00"
    lastmodifier="Markku Laine"
    lastmodified="2009-02-16T10:52:21.280+02:00"
    id="50641f86-572e-4913-ade2-a8df24f16158"
    download="http://localhost:8080/tutorial/
    selectfilesbyroles.xformsdbdownload?id=50641f86-572e-4913-ade2-
    a8df24f16158" />
  <xformsdb:file displayname="Lecture: XFormsDB" roles=""
    filename="lecture_xformsdb.pdf" mediatype="application/pdf"
    filesize="728504" comment="Held on January 27, 2009"
    creator="Markku Laine" created="2009-01-27T17:04:55.390+02:00"
    lastmodifier="" lastmodified=""
    id="f40e4173-be44-4b73-889a-2ca3ee3ae3bb"
    download="http://localhost:8080/tutorial/
    selectfilesbyroles.xformsdbdownload?id=f40e4173-be44-4b73-889a-
    2ca3ee3ae3bb" />
</xformsdb:files>
```

Listing 33: Example of use of the *file* request for uploading files

```
<!-- Insert files instance -->
<xforms:instance id="insert-files-instance">
  <xformsdb:insert>
    <xformsdb:file displayname="" roles=""
      filename="" mediatype="" filesize=""
      comment="" creator="" />
    <xformsdb:file displayname="" roles=""
      filename="" mediatype="" filesize=""
      comment="" creator="" />
  </xformsdb:insert>
</xforms:instance>

<!-- Select/Update/Insert/Delete files request instance -->
<xformsdb:instance id="suid-files-request-instance">
  <xformsdb:file />
</xformsdb:instance>

<!-- Insert files request submission -->
<xformsdb:submission id="insert-files-request-submission"
  replace="instance" instance="insert-files-instance"
  requestinstance="suid-files-request-instance" filetype="insert"
  attachmentinstance="insert-files-instance">
  <!-- Actions defined by the form author -->
  <xforms:action ev:event="xformsdb-request-error">
    <!-- Show an error message -->
    <xforms:toggle case="error-insert-files-case" />
  </xforms:action>
  <xforms:action ev:event="xforms-submit-done">
    <!-- Show a success message -->
    <xforms:toggle case="success-insert-files-case" />
  </xforms:action>
</xformsdb:submission>
```

Listing 34: Example of an XML response of the *file* request containing the metadata about the uploaded files

```
<?xml version="1.0" encoding="UTF-8"?>
<xformsdb:insert inserted="50641f86-572e-4913-ade2-a8df24f16158
  f40e4173-be44-4b73-889a-2ca3ee3ae3bb">
  <xformsdb:file displayname="API for XFormsDB 1.0" roles=""
    filename="api_for_xformsdb_1-0.pdf" mediatype="application/pdf"
    filesize="144941" comment="Editor's Draft"
    creator="Markku Laine" created="2009-01-27T17:04:55.310+02:00"
    lastmodifier="" lastmodified=""
    id="50641f86-572e-4913-ade2-a8df24f16158"
    download="http://localhost:8080/tutorial/
    selectfilesbyroles.xformsdbdownload?id=50641f86-572e-4913-ade2-
    a8df24f16158" />
  <xformsdb:file displayname="Lecture: XFormsDB" roles=""
    filename="lecture_xformsdb.pdf" mediatype="application/pdf"
    filesize="728504" comment="Held on January 27, 2009"
    creator="Markku Laine" created="2009-01-27T17:04:55.390+02:00"
    lastmodifier="" lastmodified=""
    id="f40e4173-be44-4b73-889a-2ca3ee3ae3bb"
    download="http://localhost:8080/tutorial/
    selectfilesbyroles.xformsdbdownload?id=f40e4173-be44-4b73-889a-
    2ca3ee3ae3bb" />
</xformsdb:insert>
```

Listing 35: Example of use of the *cookie* request

```
<!-- Cookie instance -->
<xforms:instance id="cookie-instance">
  <dummy xmlns="" />
</xforms:instance>

<!-- Cookie request instance -->
<xformsdb:instance id="cookie-request-instance">
  <xformsdb:cookie />
</xformsdb:instance>

<!-- Cookie request submission -->
<xformsdb:submission id="cookie-request-submission"
  replace="instance" instance="cookie-instance"
  requestinstance="cookie-request-instance">
  <!-- Actions defined by the form author -->
  <xforms:action ev:event="xformsdb-request-error">
    <!-- Show an error message -->
    <xforms:toggle case="error-cookie-case" />
  </xforms:action>
  <xforms:action ev:event="xforms-submit-done">
    <!-- Show an empty message -->
    <xforms:toggle if="string-length(
      instance( 'cookie-instance' ) ) = 0"
      case="empty-cookie-case" />
    <!-- Show a success message -->
    <xforms:toggle if="string-length(
      instance( 'cookie-instance' ) ) &gt; 0"
      case="success-cookie-case" />
  </xforms:action>
</xformsdb:submission>
```

Listing 36: Example of an XML response of the *cookie* request indicating that cookies are enabled on the browser

```
<?xml version="1.0" encoding="UTF-8"?>
<xformsdb:cookie>JSESSIONID</xformsdb:cookie>
```

Listing 37: Example of an XML response of the *cookie* request indicating that cookies are *not* enabled on the browser

```
<?xml version="1.0" encoding="UTF-8"?>
<xformsdb:cookie />
```

Listing 38: Definition of the *xformsdb:submission* element

```
<!-- XFormsDB request submission -->
<xformsdb:submission id="xformsdb-request-submission"
  replace="instance" instance="xforms-instance"
  requestinstance="xformsdb-request-instance">
  <!-- Actions defined by the form author -->
</xformsdb:submission>
```

Listing 39: Example of a detailed error (appended) from an XFormsDB implementation

```
<!-- Cookie request instance -->
<xformsdb:instance id="cookie-request-instance">
  <xformsdb:cookie>
    <!-- Detailed error (appended) from the XFormsDB
         implementation -->
    <xformsdb:error>
      <xformsdb:code>33001</xformsdb:code>
      <xformsdb:description>
        Failed to check browser support for cookies.
      </xformsdb:description>
    </xformsdb:error>
  </xformsdb:cookie>
</xformsdb:instance>
```

Listing 40: Example of a detailed error (an XHTML document) from an XFormsDB implementation

```
<?xml version="1.0" encoding="UTF-8"?>
<html>
  <head>
    <title>XFormsDB Error</title>
  </head>
  <body>
    <h1>XFormsDB Error</h1>
    <p><b>Error code:</b> 33001</p>
    <p><b>Error description:</b> Failed to check browser support for
cookies.</p>
  </body>
</html>
```

Listing 41: Definition of the *xformsdb:secview* elements for showing/hiding the part of a Web page

```
<!-- XFormsDB security view for non-logged in users -->
<xformsdb:secview>
  <!-- Actions and/or UI defined by the form author -->
</xformsdb:secview>
<!-- XFormsDB security view for logged-in users having the roles:
     user AND admin, but not having the role: moderator -->
<xformsdb:secview allroles="user admin" noroles="moderator">
  <!-- Actions and/or UI defined by the form author -->
</xformsdb:secview>
```

Listing 42: Definition of the *xformsdb:include* element for including a navigation

```
<!-- Include navigation -->
<xformsdb:include resource="xinc/navigation.xinc" />
```

Listing 43: Definition of the structure of the *xformsdb_users.xml* document

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <xformsdb:users>
    <xformsdb:user username="administrator"
      password="1224CFEC67D8695D4C9AD7B5777E61C4"
      roles="user admin" />
    <xformsdb:user username="student"
      password="1CA18BE3E542DA8D52A9E5B4E4931FC3"
      roles="user" />
  </xformsdb:users>
</root>
```

Listing 44: Definition of the structure of the xformsdb_files.xml document

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <xformsdb:files>
    <xformsdb:file displayname="API for XFormsDB 1.0" roles=""
      filename="api_for_xformsdb_1-0.pdf"
      mediatype="application/pdf"
      filesize="144941" comment="Editor's Draft"
      creator="Markku Laine" created="2009-01-27T17:04:55.310+02:00"
      lastmodifier="Markku Laine"
      lastmodified="2009-02-16T10:52:21.280+02:00"
      id="50641f86-572e-4913-ade2-a8df24f16158" />
    <xformsdb:file displayname="Lecture: XFormsDB" roles=""
      filename="lecture_xformsdb.pdf"
      mediatype="application/pdf"
      filesize="728504" comment="Held on January 27, 2009"
      creator="Markku Laine" created="2009-01-27T17:04:55.390+02:00"
      lastmodifier="" lastmodified=""
      id="f40e4173-be44-4b73-889a-2ca3ee3ae3bb" />
  </xformsdb:files>
</root>
```

Appendix B

Decision Tree Diagram of the *xformsdb:secview* Element

APPENDIX B: DECISION TREE DIAGRAM OF THE XFORMSDB:SECVIEW ELEMENT

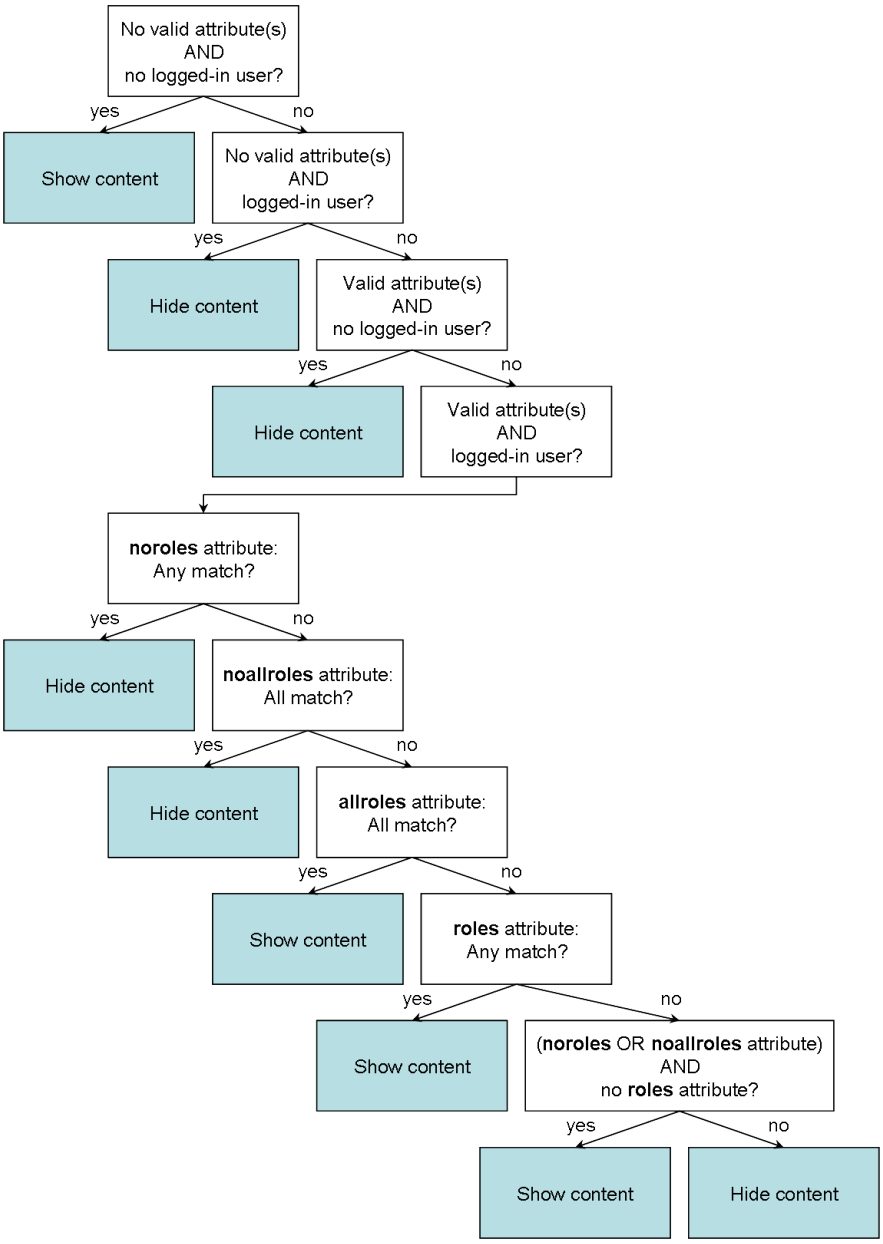

Figure 24: Decision tree diagram of the *xformsdb:secview* element

Appendix C

XFormsDB Packages

Table 33: XFormsDB packages

Package	Description
fi.tkk.tml.xformsdb.core	Provides constants used throughout the application.
fi.tkk.tml.xformsdb.dao	Provides interfaces and classes for accessing data stored in various data sources.
fi.tkk.tml.xformsdb.error	Provides classes for dealing with errors.
fi.tkk.tml.xformsdb.handler	Provides classes for handling various tasks, such as writing responses.
fi.tkk.tml.xformsdb.manager	Provides classes for managing session related data.
fi.tkk.tml.xformsdb.merger	Provides the class for performing the three-way merging of XML documents.

fi.tkk.tml.xformsdb.resource	Provides DTD, entity, XQuery expression, and XSLT transformation resources used within the application.
fi.tkk.tml.xformsdb.servlet	Provides the main servlet class of the application.
fi.tkk.tml.xformsdb.tdm	Provides classes for overriding some classes defined within the 3dm.jar package.
fi.tkk.tml.xformsdb.transformer	Provides classes for transforming data defined within XHTML+XFormsDB documents.
fi.tkk.tml.xformsdb.util	Provides utility classes.
fi.tkk.tml.xformsdb.xml	Provides classes for dealing with XML documents.