XFormsDB: An Extensible Web Application Framework
Built upon Declarative W3C Standards

Markku Laine, Denis Shestakov, Petri Vuorimaa
Department of Media Technology, Aalto University
P.O. Box 15500, FI-00076 Aalto, Finland
{markku.laine, denis.shestakov, petri.vuorimaa}@aalto.fi

ABSTRACT
Most Web applications are based on a conventional three-tier architecture, in which the presentation, application logic, and data management are developed and maintained in separate tiers. The main disadvantage of this architecture is that it requires expertise in multiple programming languages, programming paradigms, and data models used in each tier. A single expert rarely masters all the technologies and concepts involved. In this paper, we introduce a tier-expanding architectural approach that unifies the client-side (presentation tier) and server-side (logic and data tiers) programming under a single model. We base our approach on a W3C-standardized client-side markup language, XForms, and its server-side extension proposed in this paper. We derive the extension requirements from the literature and use cases, and demonstrate their functionality on the example of a blog Web application. We also show how the extension can be implemented as part of a comprehensive Web application framework called XFormsDB. The XFormsDB framework is an extensible Web application framework built upon declarative W3C standards. It has four major advantages: (1) one programming language, (2) one data model, (3) based on W3C-standardized declarative markup, and (4) extensibility in all tiers. Our conclusion is that expanding the presentation tier to cover both application logic and data management functionality makes both the development and maintenance of small- and medium-sized Web applications easier.

1. INTRODUCTION
The constantly evolving Internet has grown from an information dissemination platform to the medium running variety of applications and services. Highly interactive data-driven Web applications—commonly known as Rich Internet Applications (RIAs) [9]—are now an integral part of our lives: we use them to pay our bills, work collaboratively with our colleagues, check weather conditions, play games, browse friends’ photos, and blog about our experiences.

While the widespread adoption of RIAs has significantly improved the utility and user experience of the Web, developing such applications has dramatically increased in complexity. We can see this trend by examining Web applications based on a conventional three-tier Web application architecture [1]. In those applications, the structure and layout of a user interface is typically authored in HTML and CSS, whereas JavaScript handles the interaction. The server-side application logic, on the other hand, is implemented using an object-oriented or scripting language, such as Java, Ruby, or PHP. The client-server communication is handled using the HTML, XML, or JSON formats and asynchronous submissions, and the application data is managed with SQL statements. In addition, data-mapping libraries may be used for translating data from one format to another when moving data between different tiers. In this way, RIA developers not only need to know a multitude of systems, frameworks, best practices and languages, but also to deal with their conceptual dissimilarities [29]—indeed, the same Web application often consists of components written in imperative (e.g., Java and JavaScript) and declarative (e.g., CSS, HTML, and SQL) languages.

Unifying the client-side (presentation tier) and server-side (logic and data tiers) programming under a single model can simplify the Web application development and particularly reduce the skill set required from a developer. Reducing the number of technologies involved also makes an application more secure, as in general each technology is one more compromise in the overall application security. Generally, a unified model can be based on either server-side or client-side concepts. For instance, Google Web Toolkit (GWT)\(^2\) realizes a server-side approach, in which a general-purpose programming language—namely, object-oriented imperative Java—is used to author the application logic both on the client and the server. GWT also allows authoring the Web application user interface portion in Java. However, because the user interface design and implementation almost always require

---

1 This work is based on an earlier work: SAC ‘12 Proceedings of the 2012 ACM Symposium on Applied Computing, Copyright 2012 ACM 978-1-4503-0857-1/12/03. http://doi.acm.org/10.1145/2245276.2245407.

human involvement and judgment, while most of the server-side application logic can be covered by generic components, an approach based on a client-side programming language is a more compelling alternative.

Since our primary target group is users who are involved in Web content authoring and possess no or little programming skills, a declarative markup language is a proper candidate for this client-side based Web programming model. Thus, we chose XForms [4], an XML-based Web user interface language standardized by W3C. XForms addresses the most common problems found in HTML forms (e.g., dependency on imperative scripting languages, such as JavaScript) and eases dynamic Web form authoring by using declarative markup.

In this paper, we propose a tier-expanding architectural approach, which allows using the declarative XForms markup language on all three tiers of a Web application. We define the requirements for extending the language with common server-side and database-related functionality and present the design of the extension. We also introduce XFormsDB, a comprehensive Web application framework that supports the XForms markup language and its server-side extension proposed in this paper. Furthermore, we describe the architecture of the XFormsDB framework, present its implementation in detail, and show how it leverages declarative W3C standards on different tiers. Finally, we argue that the framework could significantly simplify the development and maintenance work of small- and medium-sized Web applications as well as reduce the skill set required from a developer. We developed the XFormsDB framework as an open source project and made it available under the MIT license. The presentation of the framework can be found elsewhere [22].

The rest of the paper is organized as follows. The next section reviews the literature relevant to this research. Section 3 provides the fundamentals of XForms and describes how the technology fits into the conventional three-tier Web application architecture. Then, an overview of XQuery is given in Section 4. In Sections 5 and 6, we present our approach of extending a client-side Web programming language with server-side functionality along with the use cases and requirements for the proposed language extension. In Section 7, we describe the design of the server-side and database-related functionalities to be included into standard XForms. Section 8 presents the implementation details of XFormsDB, a framework supporting the proposed language extension. Finally, we discuss the feasibility of our approach in Section 9 and present our conclusions and suggestions for future work in Section 10.

2. RELATED WORK
An overview of software development methodologies used for Web application development can be found in [5]. Toffetti et al. [36] reviewed the current state-of-the-art in RIA development approaches. They indicated that the current framework-based RIA development practices lack support for complete application development (client-side and server-side application logic, client-server communication, and interaction). We addressed this issue by proposing a framework based on a unified development model.

In general, unified Web application development frameworks can be based either on programming, modeling, or markup languages. For instance, Hop is a general-purpose Web programming language primarily designed for programming small- and medium-sized interactive Web applications [34, 35]. Hop—with its Scheme-based syntax—exposes a model based on two computation levels: while the application logic is executed on the first level, the second level is responsible for the graphical user interface (GUI). Though Hop separates the application logic from the GUI, it packages them together and supports strong collaboration between them via execution engines. While Hop provides an extensive set of libraries for Web development, its main drawback is not relying on any W3C-standardized language.

Hanus and Kluß presented a declarative multi-paradigm programming language called Curry [11] to describe graphical user interfaces for desktop applications as well as Web user interfaces for standard Web browsers. Curry has a Haskell-like syntax and divides the design of a user interface (UI) into three parts: structure (the hierarchical structure of UI elements, such as text inputs or select fields), functionality (the interaction with UI elements), and layout (the visual appearance of UI elements). In comparison with XForms, Curry does not rely on Web standards, and thus is unlikely to be attractive to Web developers. In addition, Curry does not provide any server-side or database-related functionality required in most Web applications.

Kuuskeri and Mikkonen [19] introduced a JavaScript-based middleware platform, extending the JavaScript language with server-side functionality. As in our approach, this proposed Web development model uses one client-side language only. The two approaches differ on the conceptual level: while Kuuskeri and Mikkonen presented a server-side language extension to imperative JavaScript, we expanded the scope of declarative XForms.

Hilda [38] is a Web application development framework based on Unified Modeling Language (UML). The data model is relational, and thus query and update operations use SQL. Hilda’s main constructs are AUnits, which correspond to UML classes. However, the presentation layer is based on HTML. The Hilda compiler translates a Hilda program into executable code: a server-side Java Servlet and client-side scripts. Compared to Hilda, XForms simplifies Web application development, since it includes both the data model and the client-side application logic. In addition, the embedding of XForms into a host markup language (e.g., XHtml) is well defined.

Unified Web application development frameworks based on markup languages are usually built either upon XML or HTML (including HTML5). Cardone et al. [6] proposed a programming model, which simplifies the design of form-based Web applications by separating client-side XML markup from the server-side programming language considerations. They based their approach on XForms, separating the data representation used on the client (XML) from the programming language structures (Java) native to the server. Unlike Cardone et al., we propose a server-side language extension to XForms that together allow developing a Web application user interface and all of its application logic using XForms only.

Hemel and Visser [13] introduced mobl, a high-level, declarative language for programming mobile Web applications. Mobl integrates languages for user interface design, styling, data modeling, querying, and application logic. Before deployment, mobl compiler translates the language into a combination of HTML, CSS, and JavaScript. Compared to XFormsDB, mobl’s approach is based more on imperative languages. Furthermore,
3. XFORMS

XForms [4], a W3C recommendation since October 2003, is an XML-based client-side forms technology and the successor to HTML forms. In contrast to conventional HTML forms, an XForms form cleanly separates the presentation, XForms User Interface, from the logic, XForms Model, and data, Instance Data, of a form by internally following the Model-View-Controller (MVC) architecture [17]. Figure 1 illustrates the main components of an XForms form.

Instance Data defines an arbitrary XML document template (the Model part of MVC) for the data to be collected in a form. The initial content and structure of an XML document can be dynamically modified afterwards through user interactions.

XForms Model uses XML to define the non-visual portion—that is, the data and the client-side application logic (the Controller part of MVC)—of a form. The data portion contains one or more Instance Data definition(s), whose structures and data types can be defined using XML Schema [8]. The logic portion embodies data submission definitions and Model Item Properties (MIPs) written in XPath [2]. The MIPs define dynamic calculations and constraints on Instance Data nodes (e.g., dependencies between various Instance Data nodes), which are impossible to define using XML Schema.

XForms User Interface provides a standard control set to declaratively define the visual portion of a form. The form controls (the View part of MVC) are bound to Instance Data nodes, allowing the separation between presentation and data.

XForms Submit Protocol defines how XForms sends and receives Instance Data as well as the serialization of that data. The data is typically transferred to and from a server, but XForms also allows saving the data to local files, for later reuse.

XForms itself does not define a document format, and therefore must always be embedded into a host language, such as XHTML or SVG. XForms also integrates seamlessly with other declarative W3C standards, including XPath (querying), XML Schema (validation), and CSS (styling). Furthermore, using XForms for authoring dynamic forms does not preclude the use of imperative scripting languages, such as JavaScript, but they can co-exist and interact within the same document.

Currently, only experimental browsers such as X-Smiles3 [15] support XForms natively. Fortunately, several options are available, ranging from browser plug-ins and client-side XSLT transformations to Ajax-based server-side transformations, that allow XForms to be used in all modern Web browsers.

3.1 Extending XForms

Conventional HTML forms offer limited extensibility options, whereas XForms has been explicitly designed from the start with extensibility in mind. The different options available for extending XForms include script, new data types and libraries, XPath extension functions, new form controls, XForms Actions, custom events, and new serialization formats [7]. However, the use of certain XForms extension options does not suit well to be used with XForms implementations relaying on native browser support or browser plug-ins because it requires end users to update the client (XForms processor) running in the browser. XForms also allows foreign attributes in all XForms elements. Foreign elements from any namespace other than XForms, however, can only be used when defined within the extension element or in a host language.

3.2 XForms in Web Applications

Figure 2 depicts the conventional three-tier Web application architecture using XForms. The architecture follows the MVC design pattern by distinctively separating the declaratively defined user interface from the application logic and persistent data residing on the server. The user interface consists of documents, which are written in XForms and use XHTML as a host language. The difference between user interface technologies used in today’s Web applications is that declarative XHTML is combined with declarative XForms instead of imperative JavaScript. The application logic portion residing on the server, on the other hand, contains custom software components using an application-specific programming language and data model. Typically, the server also hosts a server-side XForms processor in order to ensure cross-browser compatibility. The communication between the client and the server occurs asynchronously over HTTP(S), in which collected form data, Instance Data, is serialized in XML and submitted to the server using an HTTP(S) POST request. The benefit of using asynchronous submissions, similarly as in Ajax [10], is that it allows the user interface to remain responsive, while the request is being processed on the server. Finally, the server returns an XML response to the client, and the user interface is dynamically updated according to that response.

3 X-Smiles, http://www.x-smiles.org/
4. XQUERY

XQuery [3] is a declarative query language designed by W3C for extracting and manipulating data from XML documents or any data source that can be viewed as XML, such as relational databases. XQuery has a lot in common with XPath [2], another W3C recommendation for addressing parts of an XML document, as XQuery 1.0 is a superset of XPath 2.0 and they both share the same data model as well as the same set of functions and operators.

XQuery overcomes the limitations of XPath (e.g., lack of grouping, sorting, and cross document joins) by providing an SQL-like feature called a FLWOR expression, in which FLWOR stands for “for, let, where, order by, and return”, the keywords used in the expression. By using FLWOR expressions, it is possible to select and filter XML data based on specific criteria as well as transform and restructure the XML data into another XML vocabulary or structure.

One of the main design goals of XQuery was that it would use and share appropriate W3C standards as much as possible, such as XML (modeling), Namespaces (qualifying), XPath (querying), and XML Schema (validation). In addition, there are several peripheral W3C standards and working drafts that complement XQuery with capabilities, such as updating, full-text searching, and scripting. These complementary specifications, along with existing XQuery extensions, turn XQuery into a general-purpose programming language, powerful enough to replace proprietary server-side programming languages, such as Java. XQuery is widely implemented and supported by native XML databases as well as all major database vendors.

5. EXPANDING THE PRESENTATION TIER

Although this architectural change from JavaScript to XForms simplifies the development process from a Web designer’s point of view, there are still significant architectural hurdles to overcome in developing entire Web applications. For example, as Figure 3a shows, in a typical Web application using XForms in conjunction with XHTML, the server-side application logic is implemented using an object-oriented imperative language, such as Java, Ruby, or PHP. The client and the server communicate using declarative formats (e.g., XML or JSON) and asynchronous submissions. In addition, a data-mapping library for translating the data between distinct formats used on the two tiers may be used. Finally, on the undermost tier of the application, i.e., the data tier, either an ORM library or declarative SQL statements manage the data stored in a relational database.

To accomplish all of the aforementioned processes requires tier-specific experts because the programming languages, programming paradigms, and data models differ on each tier. In addition, the manual partitioning of a Web application between the client (presentation tier) and the server (logic and data tiers) complicates the development process. [18, 37]

From a Web designer’s point of view, one way of simplifying the Web application architecture is expanding the presentation tier to cover all three tiers. This presentation-centric architectural expansion allows using a single programming language and paradigm—namely, declarative XForms—as well as the XML data model throughout the entire Web application. Figure 3b depicts this presentation-centric architectural approach for extending XForms with common server-side and database-related functionality. The approach follows the MVC design pattern, where XForms can be seen as the View part, its extension as the Controller part, and XPath (part of XForms) as the language for managing the Model part stored in a database.

6. RESEARCH PROBLEM AND SCOPE

This paper explores how to extend the XForms markup language with common server-side functionality. The server-side language extension primarily aims to simplify the development and maintenance work of highly interactive data-driven Web applications so that users—mainly Web designers—can implement simple yet useful Web applications quickly and easily using only markup languages. Because most of common server-side functionality relates to data management, researching how to seamlessly integrate a standardized query language with the XForms markup language is also important. Covering the functionality of complex Web applications, however, is beyond the scope of this extension, as both the XForms markup language and the server-side extension are targeted at Web designers, who do not require advanced application logic in their Web applications.

6.1 Use Cases

The following subsections describe three possible Web applications, in which the server-side language extension can be utilized.

4 Possible candidate for XForms 1.2. Specification is available at: http://www.w3.org/MarkUp/Forms/wiki/Json
6.1.1 Address Book
Address Book\(^5\) is a simple application that allows users to store, browse, and manage information about their personal contacts, such as names, addresses, phone numbers, and e-mail addresses. In addition, the list of contacts can be sorted and the language of the user interface can be changed between Finnish, Swedish, and English.

6.1.2 Blog
Blog\(^6\) is an online journal tool for publishing content, such as news, thoughts, comments, and experiences. It allows users to browse through archives, read published posts, and leave comments on the posts. The application offers necessary tools for administrators to write new posts as well as manage published posts and comments.

6.1.3 Project Management
Project Management\(^7\) is a comprehensive software that simplifies project planning, tracking, and management. The software includes sections for managing a user’s profile, browsing announcements about news and upcoming events, following projects’ deadlines and statuses, sharing documents, and reporting working hours. Functions available on each section are determined by the roles of a currently logged-in user.

6.2 Requirements
Kaufmann and Kossmann [16] listed general requirements for Web applications that cover all three tiers of a Web application, including communication requirements. From this list, only four requirements fall within the scope of the server-side language extension: persistence and database, error handling, session management and security, and modules to facilitate recurring tasks. In addition, we included two additional general requirements, which were derived from the use cases: state maintenance as well as authentication, authorization, and access control. Finally, we defined two specific requirements for the language extension: similar syntax and processing model as well as extensible architecture.

6.2.1 General Requirements
Persistence and database: A uniform API for connecting to different types of data sources must be provided. In addition, a standardized declarative query language, which is applicable across all data sources viewable as XML, must be supported.

Error handling: A method for notifying the client about errors occurred while processing a requested server-side command must be provided.

Session management and security: Managing sessions between the client and the server must be supported regardless of the browser used or its settings. In addition, documents sent to the client must neither expose nor allow the unauthorized alteration of sensitive information.

Modules to facilitate recurring tasks: A method to facilitate modularity and the reuse of ready-made components (e.g., user interface parts and queries) in Web applications must be supported.

State maintenance: A method to maintain the state in Web applications—especially a mechanism for passing state information (e.g., Instance Data) between documents—must be supported.

Authentication, authorization, and access control: A simple way to authenticate users and to handle common access control tasks must be provided.

6.2.2 Language Extension Requirements
Similar syntax and processing model: The syntax and processing model of the server-side language extension must be similar to XForms.

Extensible architecture: The architecture for the server-side language extension must provide a method to define new features—that is, server-side commands—while retaining the same processing model.

---

\(^6\) Blog, http://testbed.tml.hut.fi/blog/
\(^7\) Project Management, http://flexi.tml.hut.fi/fs/
7. DESIGN OF THE LANGUAGE EXTENSION

This section provides a high-level description of the proposed server-side language extension, which was designed to meet the research objectives and requirements presented in Section 6. The language extension specifies common server-side and database-related functionalities, which are needed to turn XForms into a comprehensive Web programming language. The following defines the functionalities provided by the language extension: definition of server-side requests, submission of server-side requests, notification about server-side errors, permission management, and reuse of code fragments.

We chose the Blog Web application (cf. Figure 4) from the use cases to demonstrate the applicability of the proposed server-side language extension, which is used as an example throughout the rest of this paper. Listing 1 shows an excerpt of the relevant portions of the application source code. The namespace URI for the language extension used in the code example is http://www.tml.hut.fi/2007/xformsdb and is bound to the prefix xformsdb. The complete description of the proposed language extension, along with syntax definitions and usage examples, is available in [20].

7.1 Definition of Server-Side Requests

Server-side requests are commands submitted to the server, where they are securely executed. They are defined within a new element, xformsdb:instance, that acts as a wrapper for all server-side requests. The benefit of using a wrapper around server-side requests is that it enables adding new features to the language without requiring any changes to the request processing model. Currently, the language extension includes definitions for the following server-side commands: maintaining state information, logging users in and out, retrieving information about a currently logged-in user, executing queries against data sources, managing files, and checking the browser support for cookies.

The demonstrated application utilizes three of the aforementioned commands: logging users in and out for authenticating blog administrators and executing queries against data sources. Lines 27-34 show an example definition of a query command for retrieving the comments of a specific blog post, identified by an external variable $postid. In this particular example, the parameterized query expression is written in XPath and defined in an external resource (cf. line 30). The query expression is executed against a single XML document (blog.xml) stored in a database, when a corresponding submission is dispatched.

7.2 Submission of Server-Side Requests

The xformsdb:submission element is a new element that can submit server-side requests that have been defined within the same document. As with the standard XForms submissions, server-side requests can also be submitted multiple times and at any point in a form’s lifetime.

The demonstrated application has multiple server-side request submission elements for submitting various commands to be executed on the server. Lines 35-49 show how the query command defined in the previous subsection can be submitted to the server, where it is securely executed against the blog.xml document stored in the database.

For triggering the submission, the standard XForms send action is used. After a successful submission, the query result extracted from the database is stored in an XForms instance element (cf. lines 23-26), whose original content is replaced with the extracted data. Finally, lines 67-70 iterate over the data within the aforementioned XForms instance element and display it in the main content area of the Blog administration user interface.

7.3 Notification about Server-Side Errors

XForms includes a set of different events (e.g., xforms-ready and xforms-submit-done), which can be caught by standard XForms event handlers (XForms Actions) using XML Events. XForms also provides a possibility to create custom events. We have extended this set of predefined events to include a new notification-type event, xformsdb-request-error, that is dispatched to indicate a failure in a server-side request submission and/or execution process. For example, the event is dispatched if an error occurs in establishing a connection to a data source or in executing a query expression. The event’s context information, i.e., the error code and description, can be accessed with the XForms event function.

In the demonstrated application, the xformsdb-request-error event is used within all xformsdb:submission elements to catch server-side errors, as shown in lines 45-48.
Listing 1. An excerpt of the application source code of the Blog administration user interface.
The XFormsDB framework is implemented in pure Java, and includes an XFormsDB processor supporting the proposed server-side language extension. The architecture of the XFormsDB framework and the XFormsDB processor is presented below as a reference implementation of the proposed language extension.

### 8.1 The XFormsDB Framework

The XFormsDB framework is a generic platform for developing and hosting Web applications based on the XForms markup language and its server-side extension, as proposed in this paper. The framework uses a set of third-party software and libraries, including the Apache Tomcat9 HTTP Web server, the eXist-db10 native XML database (NXD) [28], and the Orbeon Forms11 Ajax-based server-side XForms processor.

Figure 5 depicts the high-level architecture of the XFormsDB framework. Here, it differs from the conventional three-tier Web application architecture using XForms (cf. Figure 2) in that a generic software component (an XFormsDB processor) replaces the functionality provided by custom server-side software components. Because of this architectural change, all application development is now moved to the client side and is performed in extended XHTML+XForms documents. Because of this architectural change, all application development is now moved to the client side and is performed in extended XHTML+XForms documents. The server also hosts an Ajax-based server-side XForms processor called Orbeon Forms, which in the end—if necessary—transforms these documents into cross-browser (X)HTML+CSS+JavaScript or plain (X)HTML+CSS, depending on the configuration. The communication between the client and the server happens asynchronously over HTTP(S). Currently, the framework supports only XML-based data sources (XML documents and eXist-db) but by using a middleware, e.g., DataDirect XQuery12, support for other data sources (e.g., relational databases) can be easily added. In the demonstrated application, all data was stored in the eXist-db native XML database.

### 8.2 The XFormsDB Processor

The XFormsDB processor is a generic software component supporting the proposed server-side language extension. The processor’s responsibilities include handling requests and writing responses, transforming extended XHTML+XForms documents, managing sessions, performing synchronized updates, and providing integration services to heterogeneous data sources. Separate components carry out each of these tasks, as depicted in Figure 6.

When a client makes an HTTP(S) request to the server, the request first reaches the XFormsDB processor and is handled by its front controller, XFormsDB Servlet. The front controller extracts relevant request information and forwards the request to an appropriate request handler. In case an extended XHTML+XForms document is requested, XFormsDB Transformer processes the document according to the following steps:

---

10 eXist-db, http://exist-db.org/
1. Parse the document and identify server-side extension elements.
2. Incorporate all external documents into the main document (cf. \texttt{xformsdb:include}).
3. Filter out those parts to which the user does not have access rights (cf. \texttt{xformsdb:secview}).
4. Identify and collect information from other relevant elements (e.g., \texttt{xformsdb:instance}, \texttt{xformsdb:query}, and \texttt{xformsdb:submission}).
5. Store the collected information found in step 4 in the session (\textit{XFormsDB Managers}).
6. Transform the document (including the server-side extension elements) into XHTML+XForms 1.1 compliant markup, in which the definitions of server-side commands containing sensitive information have been substituted with opaque reference IDs for security reasons. During the transformation process, certain utility instances (e.g., an \textit{Instance Data} containing HTTP request headers) are automatically added to the document.
7. Return the transformed document.

Before returning the transformed document to the client, the document goes through another transformation process (cf. Orbeon Forms) that transforms it into a format viewable by the requesting client.

Asynchronous form submissions over HTTP(S) also go through the front controller (\textit{XFormsDB Servlet}), which extracts relevant request information and forwards the request to an appropriate request handler based on the submitted command. In the case of a \textit{query} command submission, the original query expression is fetched from the session (\textit{XFormsDB Managers}) using the opaque reference ID submitted along with the command, and then executed against the underlying data source (\textit{XML Document} and \textit{eXist-db Adapters}). Finally, a response XML is composed and returned to the client.

8.2.1 Data Synchronization
The \textit{XFormsDB} processor includes built-in support for performing synchronized updates (\textit{3DM XML 3-Way Merger}). To accomplish data synchronization, the \textit{XFormsDB} processor uses 3DM\textsuperscript{13} [23], a middleware for performing three-way merging of XML documents, which is able to detect and handle \textit{update}, \textit{insert}, and \textit{delete} operations as well as \textit{moves} and \textit{copies} of entire subtrees. Furthermore, the aforementioned operations can be performed without the use of unique element identifiers, i.e., original XML documents can be used as such without equipping them with excess attributes.

We illustrate how the 3DM merging process works in the example shown in Figure 7. In the example, (a) is referred to as the original version, (b) as the altered version, (c) as the current version stored in the data source, and (d) as the merged version. Green color indicates that the node has been either updated (marked with an asterisk), inserted, or moved, whereas white color indicates that the node has remained unaltered.

In \textit{XFormsDB}, the updating process with data synchronization includes the following steps. In the first step, an XML fragment is retrieved from a data source using an XPath expression that points to the root element of the XML fragment to be updated. Then, the retrieved XML fragment can be altered on the client, after which the altered XML fragment is submitted back to be stored in the data source using the same XPath expression as before. Next, the data synchronization process is performed and upon a successful synchronization, the result XML fragment is stored in the data source. Finally, the stored XML fragment, which may contain changes made concurrently by other clients, is returned to the client. In case the data synchronization process fails (e.g., a merge conflict), an appropriate error message is reported back to the client, which handles the error on a case-by-case basis.

8.3 Extensibility and Limitations
The \textit{XFormsDB} framework supports extensibility at different levels of the architecture. The most elegant way of extending the architecture is by defining new server-side requests to the language extension, as stated in Subsection 7.1. Listing 2 shows a simple example of how to define a new server-side request (cf. line 2) for retrieving HTTP request headers from a hosting server. In the case of the Blog Web application, the retrieved information could be used, for instance, to detect mobile clients and redirect them automatically to the mobile-optimized version of a particular webpage.

```
01. <xformsdb:instance id="http-request-headers">
02.  <xformsdb:httprequestheaders />
03. </xformsdb:instance>
```

\textbf{Listing 2.} An example of a new server-side request definition.

The main disadvantage of this approach is that every syntax addition made to the language extension also needs to be implemented in the \textit{XFormsDB} processor. In its current state, extended XHTML+XForms documents provide the basic means for users to implement simple yet useful Web applications using only markup languages. Due to the limited expressive power of the languages, however, they are alone insufficient to meet the requirements of more complex Web applications. In the following subsections, we give examples of how the expressive power of the

\textsuperscript{13} 3DM, http://developer.berlios.de/projects/tdm/
XFormsDB framework can be improved by using different technologies specific to each tier.

8.3.1 Presentation Tier
The XFormsDB framework relies mainly on XForms on the presentation tier. XForms uses declarative markup to define both the form controls and related client-side application logic of a Web application. It also offers a wide variety of options for extensibility, as described in Subsection 3.1.

The primary reasons for using extensions on the presentation tier include adding animations, interactivity, and client-side application logic to web pages that go beyond the capabilities of XForms. In the demonstrated application, extended XHTML+XForms documents are supplemented with imperative JavaScript embeddings for the purpose of measuring the response time of each webpage. In addition, one of Orbeon Forms’ XForms extension attributes is used to format dates into a human-readable form (cf. Listing 3).

8.3.2 Logic Tier
Our proposed server-side extension to XForms is responsible for covering common server-side and database-related functionality required by most small-sized Web applications. To meet more advanced server-side application logic requirements, XQuery (cf. Section 4), and especially eXist-db’s XQuery extension functions\(^{14}\) can be utilized. eXist-db’s functions are divided into pluggable modules and, as a whole, provide an extensive set of functionality ranging from small utility functions (e.g., for performing date and time operations as well as transforming XML into JSON) to complete libraries, such as the HTTP Client module.

In the Blog Web application, we did not resort to any eXist-db’s XQuery extension modules. Nonetheless, we demonstrate an example of how an extension module could be used in XQuery code (cf. Listing 4).

Listing 3. An example of Orbeon Forms’ XForms extension attribute used in the xforms:output element.

```
01. |<xforms:output ref="creationTime">
02.  |xxforms:format="format-date( xs:date( . ),
03.  |'[Mn] [D], [V]', 'en', (), () )" />
```

Listing 4. An example of eXist-db’s XQuery extension module used in XQuery code.

```
01. |xquery version "1.0" encoding "UTF-8";
02. |declare namespace httpclient =
03. |"http://exist-db.org/xquery/httpclient";
04. |...
05. |httpclient:get( $url, $persist, $requestHeaders )
```

8.3.3 Data Tier
As described in Subsection 8.2.1, the XFormsDB framework provides a simple and elegant means for performing queries and synchronized updates using XPath only. Though this approach has its advantages, it also inherits few problems to be addressed. For instance, performing a simple \textit{insert} or \textit{delete} operation requires redundant transfer of large XML fragments between the client and

\(^{14}\) eXist-db’s XQuery extension modules, http://exist-db.org/exist/extensions.xml
the server. In addition, an XML fragment that needs to be updated might expose sensitive information to the client.

To overcome the limitations of the aforementioned method, and XPath in general, the XFormsDB framework provides an option to use a more expressive query language, XQuery. The demonstration application relies heavily on XQuery and its standard functions as well as the functions defined in the FunctX XQuery Function library\(^{15}\). Listing 5 shows an XQuery expression which retrieves a single blog post identified by the external variable $id$ and returns a custom XML document as a response.

```
01. xquery version "1.0" encoding "UTF-8";
02. declare namespace xformsdb = "http://www.xml.tkk.fi/2009/xformsdb";
03. declare variable $id as xs:string external;
04. for $post in /root/blog/posts/post
05. where $post/@id = $id
06. return
07. <post id="{$post/@id}">
08. <headline>{$post/headline/text()}</headline>
09. <creationtime>{$post/creationtime/text()}</creationtime>
10. <content>{$post/content/text()}</content>
11. <author>{$post/author/text()}</author>
12. <comments>
13. count($post/comments/comment)
14. </comments>
15. </post>
```

Listing 5. An XQuery expression for retrieving a specific blog post identified by its $id$.

Besides the extension methods discussed above, the XFormsDB framework allows the use of the same XML Schema document both on the client and the server to validate the structures and data types of transmitted XML documents. Furthermore, with the extension methods, the framework becomes fully compatible with the XRX (XForms/REST/XQuery) architecture\(^{26}\), making it a viable option for developing complex Web applications.

An obvious drawback of each extension method is, however, that they all require a user to learn a new technology. The framework also shares some of the problems that are common to many Ajax-based Web applications, such as problems related to the use of the browser’s back button and bookmarking. In addition, the XFormsDB framework assumes that the user possesses a basic knowledge of XForms and our proposed server-side language extension, which may be a barrier for Web content authors. Fortunately, these limitations can be addressed by providing a Web-based tool that allows authors to visually develop XFormsDB-based Web applications. The prototype of such a tool called XFormsDB IDE (XIDE)\(^{16}\) has already been implemented and the results have been published in a separate paper\(^{24}\).

9. DISCUSSION

Typically, the amount of code in Web applications is distributed approximately equally between the client and the server\(^{16}\). In the Blog Web application that uses the XFormsDB framework, this ratio was 90% and 10% respectively\(^{20}\), meaning that our proposed server-side language extension (including XPath and XQuery code) significantly reduced the amount of code required to develop the server-side application logic and data management functionalities. In terms of lines of code (LoC), this means that the amount of code required for implementing the Blog Web application was decreased by 45%—that is, approximately 2400 LoC\(^{17}\). Detailed metrics for the Address Book and Blog Web applications (cf. Sections 6.1.1 and 6.1.2, respectively) are available in\(^{20}\).

The advantages of using the XRX architecture compared to the conventional three-tier architecture are further discussed in\(^{31}\). In their paper, Nemeć et al. show that applications developed according to the XRX architecture are more efficient and elegant. They continue by stating that the XRX architecture increases productivity and reduces implementation costs.

According to Cardone et al.\(^{6}\), there are three main reasons for the inefficiency of the conventional three-tier architecture when it comes to developing complex Web applications. First, dynamic webpages are often generated on the fly, making application source code harder to understand and debugging more difficult. Second, dynamic webpages often contain a mixture of markup languages, client-side scripting code, and server-side function calls, making 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.

Using only one language on all three tiers reduces the number of technologies involved and can unify the Web development process\(^{21}\). Determining the most suitable language for building Web applications thus becomes a question. According to Schmitz\(^{33}\), declarative languages (e.g., XHTML) have several advantages over imperative languages (e.g., Java). Particularly, one compelling advantage is that most Web content authors, not being programmers, prefer declarative languages. Moreover, content authors working on the presentation tier are, as a rule, familiar with declarative (X)HTML and CSS, but not the server-side aspects of a Web application. They can thus benefit from a client-side language that has been extended with server-side functionality.

To justify the choice of our client-side programming language, we followed a recent survey\(^{32}\), in which five XML-based client-side languages, including HTML5\(^{14}\) and XForms, were evaluated. According to the study, XForms is best suited for data-intensive applications and applications with accessibility requirements. XForms also provides a rich declarative use of client-side data and can easily define interdependencies between the data and user interface.

---

\(^{15}\) FunctX XQuery Function Library, http://www.xqueryfunctions.com/

\(^{16}\) XFormsDB IDE (XIDE), http://code.google.com/p/xformsdb-ide/
10. CONCLUSIONS AND FUTURE WORK
In this paper, we addressed the challenge of simplifying the development and maintenance work of highly interactive data-driven Web applications. We proposed a presentation-centric architectural approach that allows users to implement simple yet useful Web applications using only markup languages. The approach is based on a W3C-standardized client-side technology, the XForms markup language, and its server-side extension proposed in this paper. In addition, we presented the XFormsDB framework, a comprehensive implementation of the proposed approach and the language extension. The framework has four major advantages. First, the entire application can be developed using a single markup language and within a single document (i.e., an extended XHTML 2.0 document). Second, the same data model (i.e., XML) can be used across all tiers of a Web application. Third, the approach is based on declarative programming, and thus allows users with limited programming skills (particularly Web user interface designers who are well familiar with declarative HTML and CSS) to create entire Web applications. Finally, the framework offers a variety of options for extensibility, mainly by leveraging declarative W3C standards. This framework, together with a number of examples, is available under the MIT license at http://code.google.com/p/xformadb/.

In our future work, we will mainly focus on the next two functionality aspects: (1) real-time communication and (2) client-side storage. By adding real-time communication capabilities (namely, XMPP over WebSocket [30] together with a declarative API definition) to the XFormsDB framework, opens up new possibilities for developing more dynamic, event-driven Web applications. To adapt the XFormsDB framework to the requirements of mobile Web application development, we plan to extend our framework with the support for client-side databases (e.g., IndexedDB [27]). Together these technologies can yield significant improvements in performance and user experience for highly interactive data-driven Web applications.

11. ACKNOWLEDGMENTS
This research work was conducted as part of TIVIT’s Flexible Services program and its Ecosystem Design and Evolution (EDEN) project. The funding for this project was granted by Tekes and Nokia Research Center.

12. REFERENCES


ABOUT THE AUTHORS:

Markku Laine is a doctoral student in the Department of Media Technology at the Aalto University, Finland. He received his M.Sc. degree in Communications Engineering from the Aalto University in 2010. His current research interests include declarative Web application development, Web performance optimization, and real-time communication protocols. Contact him at markku.laine@aalto.fi.

Denis Shestakov is a postdoctoral researcher in the Department of Media Technology at the Aalto University, Finland. He earned his Ph.D. (2008) degree from the University of Turku, Finland. His dissertation addressed the limitations of Web crawlers, specifically the poor coverage of information available in online databases (a.k.a. the Deep Web). His current research interests span the areas of distributed data management and big data processing, with a particular focus on scalable Web agents and services. Contact him at denis.shestakov@aalto.fi.

Petri Vuorimaa is a full professor in the Department of Media Technology at the Aalto University, Finland. He obtained both his M.Sc. (1990) and D.Sc. (1995) degrees from the Tampere University of Technology, Finland. His current research interests include Web-based services, smart spaces, and mobile media applications. Contact him at petri.vuorimaa@aalto.fi.