World Wide Web (2016) 19:519–543
DOI 10.1007/s11280-015-0339-z

Leveraging declarative languages in web application development

Petri Vuorimaa · Markku Laine · Evgenia Litvinova · Denis Shestakov

Received: 7 February 2014 / Revised: 24 February 2015 / Accepted: 4 March 2015 /
Published online: 2 April 2015

Abstract Web Applications have become an omnipresent part of our daily lives. They are easy to use, but hard to develop. WYSIWYG editors, form builders, mashup editors, and markup authoring tools ease the development of Web Applications. However, more advanced Web Applications require servers-side programming, which is beyond the skills of end-user developers. In this paper, we discuss how declarative languages can simplify Web Application development and empower end-users as Web developers. We first identify nine end-user Web Application development levels ranging from simple visual customization to advanced three-tier programming. Then, we propose expanding the presentation tier to support all aspects of Web Application development. We introduce a unified XForms-based framework—called XFormsDB—that supports both client-side and server-side Web Application development. Furthermore, we make a language extension proposal—called XFormsRTC—for adding true real-time communication capabilities to XForms. We also present XFormsDB Integrated Development Environment (XIDE), which assists end-users in authoring highly interactive data-driven Web Applications. XIDE supports all Web Application development levels and, especially, promotes the transition from markup authoring and snippet programming to single and unified language programming.

Keywords Web framework · Web application · Web development · End-user development · Declarative language · Real-time communication

P. Vuorimaa (✉) · M. Laine · E. Litvinova · D. Shestakov
Department of Computer Science, Aalto University, Aalto FI-00076, Finland
e-mail: petri.vuorimaa@aalto.fi

M. Laine
e-mail: markku.laine@aalto.fi

E. Litvinova
e-mail: evgenia.litvinova@aalto.fi

D. Shestakov
e-mail: denis.shestakov@aalto.fi

1 Introduction

Highly interactive data-driven Web Applications—abbreviated to *Web Applications* in the following—are usually based on the so-called three-tier architecture [2]. The *presentation* tier (i.e., user interface) is defined using HTML and CSS languages, and complemented with numerous JavaScript embeddings for client-side application logic. The *logic* tier (i.e., server-side application logic) is based either on an object-oriented (e.g., Java or Ruby) or scripting (e.g., PHP) language and uses HTML, XML, or JSON formats for client–server communication. Finally, the *data* tier (i.e., application data) uses either an Object-Relational Mapping (ORM) library or SQL statements for data management.

Typically, Web Applications contain both imperative (e.g., Java and JavaScript) and declarative (e.g., HTML, CSS, and SQL) components. Thus, Web developers have to master several programming languages and face their conceptual differences [39]. A common solution is to assign tier-specific professionals (i.e., Web designers, software engineers, and database experts) to develop each tier.

On the other hand, end-users also participate in Web development. Without professional help, they create and edit Web Applications that assist them in their daily life activities [11]. The scope of their applications is wide—end-users write in forums and wikis, create mashups and surveys, edit media rich blog pages, create basic HTML pages, etc. In the following, the term *end-user developers* refers to non-professionals, who do some Web development to support their professional or leisure activities.

End-user developers often use dedicated visual tools, such as survey builders, mashup editors, or component-based tools [23] to create Web Applications. Unfortunately, these visual tools have their limitations. While professional tools are more expressive, they require more extensive knowledge of different programming languages, technologies, and paradigms. Thus, end-user developers face the complexities of professional Web Application development described above.

In this paper, we aim to bridge the gap between end-user and professional Web Application development. We focus on end-users who need to move forward from markup authoring and snippet programming. First, we analyze how Web Application development complicates when proceeding from end-user to professional Web development. At the same time, we identify the core learning barriers. Then, we discuss how end-users can leverage their existing Web development skills to overcome these learning barriers.

According to a recent review [54], there is a lack of support for complete Web Application development: server-side and client-side application logic, client–server communication, and interaction. Another recent survey [10] shows that research focuses on either model-driven design or implementation issues—a distinct gap exists between them. In this paper, we propose that a single, declarative model is used both on the client-side (i.e., presentation tier) and server-side (i.e., logic and data tiers); including client–server communication and interaction. Declarative approach is beneficial, since end-user developers often have some experience in using declarative languages. In addition, the application security improves as, in general, each technology is one more compromise to the overall application security.

Our contributions This paper discusses declarative Web Application development. Our specific contributions related to framework, IDE, and minor/other contributions are as follows:

- Nine-level classification of end-user Web Application development activities with a special focus on advanced Web development (minor)
- Presentation-centric architectural approach based on W3C-standardized declarative languages for unifying Web Application development (framework)
- Set of requirements for extending a presentation-centric declarative language with common server-side and database-related functionality; including real-time communication capabilities (framework)
- Language extension (XFormsDB) for turning XForms [5] into a comprehensive Web Application development language (framework)
- Language extension (XFormsRTC) for adding real-time communication capabilities to XForms (framework)
- Web framework (XFormsDB) implementing XForms and the XFormsDB language extension for declarative Web Application development (framework)
- Web-based visual authoring tool (XIDE) for the XFormsDB framework to assist lower-level end-user developers in Web Application development (IDE)
- Evaluation of the XFormsDB framework and the XIDE authoring tool (framework and IDE)

This paper is organized as follows. Next, Section 2 presents related work. Then, Section 3 discusses altogether nine end-user development levels. After that, Section 4 describes how the presentation tier can be expanded to cover all aspects of Web Application development. Section 5 defines requirements to extend XForms with common server-side and database-related functionality, including Real-Time Communication (RTC) capabilities. Then, Section 6 describes the proposed XFormsDB and XFormsRTC language extensions, while Section 7 discusses practical implementation issues. Next, Section 8 introduces the XIDE editor: a visual tool for end-user Web Application development. Then, Section 9 presents evaluation of both the XFormsDB framework and the XIDE editor. Finally, Section 10 contains conclusions.

2 Related work

In this Section, we present related work. Based on two recent surveys [10, 54], we identify WebML-RIA [7, 15] as being closest to XFormsDB, and thus provide a thorough comparison between these two approaches. Finally, we discuss the differences between declarative and imperative Web Application development.

2.1 Rich internet applications

According to Toffetti et al. [54] Rich Internet Application (RIA) development approaches can be divided into Code-based, Framework-based, and Model-driven methodologies. In the first approach, developers code directly using technology-specific programming languages. Frameworks contain more advanced libraries and code-generation tools. However, frameworks typically focus on client-side, and thus lack support for complete application development. Model-driven methodologies are more comprehensive and rely on automatic code-generation. However, model-driven methodologies usually have difficulties in expressing advanced RIA features.

Toffetti et al. [54] compare the different development approaches based on their technology, language, and process-related features. We evaluated our XFormsDB framework using the same criteria. The closest Framework-based methodologies are AJAX [16] libraries, AJAX code-generators, *OpenLazzlo* (http://www.openlaszlo.org/), and *Adobe Flex,*[1] while the closest Model-driven methodologies are WebML-RIA [7, 15], UsiXML [36], and OOWS for RIA [55].

Model-driven methodologies are typically declarative as XFormsDB. However, most model-driven methodologies focus on modelling the presentation and behavioral issues at client side. Only WebML-RIA [7, 15] models also data issues as XFormsDB. Most model-driven methodologies also rely on back-to-front development process. XFormsDB is based on more user-centered design as UsixML [36] and OOWS for RIA [55].

2.2 WebML-RIA

WebML-RIA [7, 15] models are composed of four sub models: domain, hypertext, dynamic, and presentation. XForms is based on conventional Model-View-Controller architecture, whose parts resemble WebML-RIA's domain, presentation, and dynamic models. However, there are major differences. In WebML-RIA, domain data model is expressed either as entity-relationships diagrams or semantic representations (e.g., RDF or OWL), while XForms uses XML Schemas. In essence, both WebML-RIA dynamic model and XForms controller describe how events are handled inside the Web Application. The XFormsRTC extends client–server communication with bidirectional real-time support. In WebML-RIA, hypertext model defines the composition and navigation of the user interface (UI), while the overlaying presentation model describes the look and feel of the UI. In XForms, the view defines the structure of the form. However, the UI is always embedded in a host language, such as XHTML. The host language defines the UI and navigation. Thus, the division is a bit different than in WebML-RIA.

WebML-RIA and XFormsDB have also implementation differences. The WebML-RIA IDE has been implemented using the WebRatio tool, while XFormsDB has its own Web-based IDE, XIDE. The WebML-RIA server-side implementation is also based on WebRatio and its Apache Struts2 framework and Hibernate persistence layer. XFormsDB server-side implementation is based on our XFormsDB processor and Orbeon Forms XForms processor. The WebML-RIA client runtime environment is OpenLazzlo, while XFormsDB uses (X)HTML+CSS+JavaScript.

2.3 Declarative vs. imperative web application development

AJAX libraries, such as *Dojo* (http://dojotoolkit.org/) and *jQuery* (http://jquery.com/), are imperative technologies, while XFormsDB is declarative. AJAX libraries typically use only declarative UIs based on HTML. AJAX code-generators are even more imperative. OpenLazzlo has also declarative UI, but is otherwise imperative. Same applies to Adobe Flex, which is based on ActionScript rather than JavaScript.

Kuuskeri and Mikkonen [26] introduced a JavaScript-based middleware platform, extending the JavaScript language with server-side functionality. As in our approach, this proposed Web development model used only one client-side language. The two approaches differ on the conceptual level: while Kuuskeri and Mikkonen [26] presented server-side extensions to imperative JavaScript, we expanded the scope of declarative XForms.

[1] Currently known as Apache Flex (http://flex.apache.org/).

In [28], we evaluated three frameworks in detail, each representing the Web tier-expansion approach: our declarative XFormsDB, imperative *Google Web toolkit (GWT)*[2], and declarative Sausalito[3] [22], which expand the presentation, logic, and data tiers, respectively. The logic-centric imperative GWT is most mature and powerful of the three frameworks. It has the best documentation and tools support and the most active development community. While the data-centric declarative Sausalito (with the declarative XQuery as a base language) provides the best scalability support, the presentation-centric declarative XFormsDB is an attractive solution for end-user developers familiar with declarative markup languages.

Using single language on all three tiers reduces the number of technologies involved and can unify the Web development process [28]. According to Schmitz [52], declarative languages (e.g., (X)HTML) have several advantages over imperative languages (e.g., Java). Especially, declarative languages are more accessible to end-user developers (i.e., non-programmers). Moreover, end-user developers are more familiar with declarative, W3C-standardized (X)HTML and CSS than server-side aspects of Web Application development. Thus, they 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 [44], in which five XML-based client-side languages—including HTML5 [18] and XForms—were evaluated. According to the study, the XForms language is best suited for data-intensive applications and applications with accessibility requirements. The XForms language also provides a rich declarative use of client-side data and can easily define interdependencies between the data and user interface.

3 End-user web application development

In practice, all the tools and frameworks discussed in the previous section are intended for professional developers. To understand better end-user developers' requirements, we analyze the evolution of Web Application development from Web surfing to professional Web development. The analysis is based on a survey of the literature and existing tools. More precisely, we expand end-user development classification described in [11]. In [32], we defined six levels of end-user Web Application development between Web surfing and professional development. In this paper, we focus especially on advanced Web Application development, and thus we deconstruct the two most advanced levels into five new levels of programming activities.

As depicted in Figure 1, we identify nine Web Application development levels. Far left is Web surfing, i.e., no Web development. Far right is professional Web Application development. Each level in between defines both the activities a user is able to perform and corresponding skills he/she needs to possess.

> **L1 Customizing components:** End-users adjust existing Web Applications by setting values to parameters, such as adjusting colors of a personal blog page.
> **L2 WYSIWYG editing:** End-users develop static Web pages or targeted applications (e.g., surveys) in What You See Is What You Get (WYSIWYG) editors. Example tools are *Google Sites* (http://sites.google.com/), *JotForm* (http://www.jotform.com/), and *Orbeon Form Builder* (http://www.orbeon.com/).

[2] Currently known as GWT (http://www.gwtproject.org/).
[3] Currently known as 28.io - Virtual Databases (http://www.28.io/).

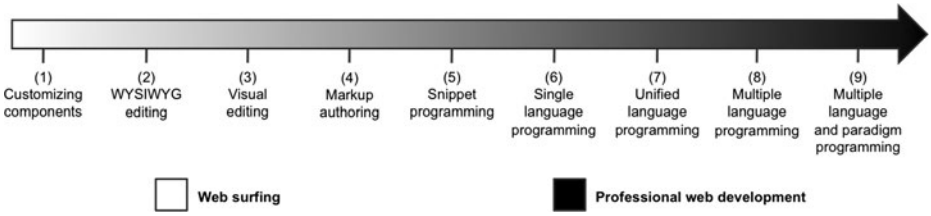

Figure 1 Levels of Web Application development

L3 Visual editing: End-users create Web Applications visually by adding, customizing, and connecting components [46]. Common examples are mashup editors, such as *Yahoo! Pipes* (http://pipes.yahoo.com/pipes/) and *DashMash* [9], and component-based editors, such as *EzWeb/FAST* [33, 56]. Visual editing is possible without programming skills and syntax knowledge. However, end-users often need to understand the tool's programming paradigm and application-creation logic.

L4 Markup authoring: End-users contribute to wikis and write rich-text posts to blogs using plain markup languages. This is a code editing activity; however, the markup languages are often simplified. Nevertheless, end-users learn the basics of declarative programming. For example, *Wikipedia* authors use an internal markup language to add their contributions to articles.

L5 Snippet programming: End-users copy-paste or manually edit existing source code [8]. For example, *Adobe Dreamweaver* (http://www.adobe.com/products/dreamweaver.html) allows end-users first to create a Web page visually in a drag-n-drop editor and then manually edit the auto-generated source code. However, end-users find switching the views between visual and source code editors difficult [42]. With *WordPress* (http://wordpress.org/) end-users can create Web Applications using predefined templates and widgets implemented in PHP, which is rarely known among the end-users at the skill level 5. In addition, end-users need to know WordPress' internal architecture and API before they can do even minor changes to widgets. *Click* [47] allows end-users to create Web Applications using component-based approach. The tool provides several layers of modification complexity, starting from customizing templates to modifying and extending the component framework and editing PHP code.

L6 Single language programming: End-users have a sufficient knowledge of a single programming language and they are able to develop a piece of functionality from scratch. However, they are unaware of other programming languages and paradigms, so they can only apply their knowledge on one tier, either for creating a static user interface, developing the server-side part of application logic, or managing data. For example, an end-user might know (X)HTML to create a static user interface. In order to create a fully interactive data-driven Web Application, end-users need to rely on other developers or tools.

L7 Unified language programming: End-users have sufficient knowledge of a single programming language or technology, which can be used to create all necessary components of a Web Application. In addition, end-users are able to manage the communication between the components. When a single programming language is used on all three tiers, end-users often use special toolkits or frameworks, which process the source code and translate it to low-level, tier-specific languages. For example, GWT allows the creation of entire Web Application using only Java language. In compilation phase, GWT automatically converts the Java source code to JavaScript and a database language.

L8 Multiple language programming: End-users know several complementary languages and can combine them. The main difference to the level 7 is that end-users need to master and integrate several languages together. However, the languages utilize the same paradigm (i.e., declarative or imperative), which makes it less demanding for end-users to combine them. For example, *eXist-db* (http://exist-db.org/) [38] combines two declarative languages (i.e., XForms [5] and XQuery [4]) to support the creation of Web Applications.

L9 Multiple language and paradigm programming: End-users are almost professional programmers. They know how to combine several complementary technologies to construct an interactive client–server Web Application. At the level 9, the main challenge is that technologies can be declarative, imperative, or even multi-paradigm. Therefore, end-users need to understand both paradigms and master the communication between different parts of a Web Application.

Currently, professional Web Application development takes place between the levels 6 and 9. Usually, a professional Web developer masters at least one programming or unified programming language, while a development team can handle multiple programming languages and paradigms necessary to implement the entire Web Application.

Although end-users are less advanced than professional developers, they can learn new skills. Their main motivation is to improve the current or new application's functionality, which requires higher-level skills and tools. As the analysis shows, the lower-level tools presuppose only basic Web Application usage skills. However, the tools restrict the scope of developed Web Applications. As end-users move to the right, they can create more versatile Web Applications, but they must master more advanced skills.

The critical transition happens between the levels 4 and 6, where end-users have to learn a programming language. However, a declarative language can lower this barrier [52]. The key ingredient is the markup language that the level 4 end-users learn at least partially. Since declarative languages—such as (X)HTML and XML—look familiar, they can do some code editing based on their existing knowledge. However, currently pure HTML supports only the creation of static Web pages. The level 5 and 6 end-users have limited programming skills, and thus they usually face difficulties in imperative server-side development.

On the levels 7 and 8, a unified programming model can be based on either server-side or client-side concepts. For example, GWT realizes the server-side approach using a general-purpose programming language (i.e., object-oriented imperative Java) to author both the server-side application logic and Web Application user interface. However, the user interface design and implementation almost always requires human involvement and judgment. Thus, we consider a full Web Application development cycle—based on a client-side programming language—as a more compelling alternative. The client-side approach is particularly feasible, since—unlike user interface and interaction behavior—most server-side functionalities can be covered by generic components. In addition lower-level end-user developers are more familiar with declarative languages.

4 Expanding the presentation tier

In this paper, we target end-user developers who author Web content, but possess limited programming skills to develop entire Web Applications on their own. As we analyzed in the previous section, many users reach the level 4, in which they learn the basics of declarative languages. Thus, a declarative markup language meets the requirements of the client-side Web

programming model. We propose to use XForms [5], which is an XML-based Web user interface language designed to tackle the most common problems found in HTML forms. In addition, XForms removes the dependency on imperative scripting languages, such as JavaScript. Thus, when used in conjunction with XHTML, it becomes possible to author dynamic Web user interfaces (i.e., presentation tier with client-side application logic) without JavaScript.

This architectural change to fully declarative user interfaces simplifies the development process. However, end-user developers still have to face significant architectural hurdles, as depicted in Figure 2a. In conventional XForms-based Web Applications, 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[4]) and asynchronous submissions over HTTP(S). On the lowest application tier (i.e., data tier), either declarative SQL statements or an ORM library manage data stored in a relational database.

The above-mentioned development 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 (i.e., presentation tier) and the server (i.e., logic and data tiers) complicates the development process. [26, 57]

From a developer's point of view, one way to simplify the Web Application development even further is to expand the presentation tier to cover all three tiers. This presentation-centric architectural expansion allows the use of XForms—as well as the XML data model—as the only programming language and paradigm throughout the entire Web Application. In addition, it removes the need for using a separate data-binding framework, such as WebSoDa [17]. Figure 2b depicts this presentation-centric architectural approach for extending XForms with common server-side and database-related functionality.

Currently, only experimental browsers such as X-Smiles (http://www.xsmiles.org/) [20] natively support XForms. Fortunately, several solutions[5]—from browser plug-ins and client-side XSLT transformations to Ajax-based server-side transformations—allow XForms to be used in all modern Web browsers. Our XFormsDB framework includes Orbeon Forms XForms processor, which automatically converts XForms documents into cross-browser (X)HTML+CSS+JavaScript documents. In addition, business logic is handled by generic server side XFormsDB processor (cf. Section 7).

5 XForms as a unified end-user web application development language

XForms [5]—a W3C recommendation since October 2003—is an XML-based client-side forms technology. In contrast to conventional HTML forms, XForms cleanly separates the presentation (i.e., XForms User Interface) from the logic (i.e., XForms Model) and data (i.e., Instance Data) with its internal Model-View-Controller (MVC) architecture [25].

Our objective is to extend the presentation-centric XForms language with common server-side and database-related functionality, as described in Section 4. In addition, we extend XForms with two-way, full-duplex, true RTC capabilities. These language extensions support transition from the level 6 to the level 7, and thus enable end-user developers to develop entire Web Applications on their own. In addition, development and maintenance of Web Applications becomes easier as authoring is done using only markup languages. Because most of

[4] JSON support is planned for XForms 2.0 [6].
[5] List of XForms implementations is available at http://www.w3.org/community/xformsusers/wiki/XForms_Implementations.

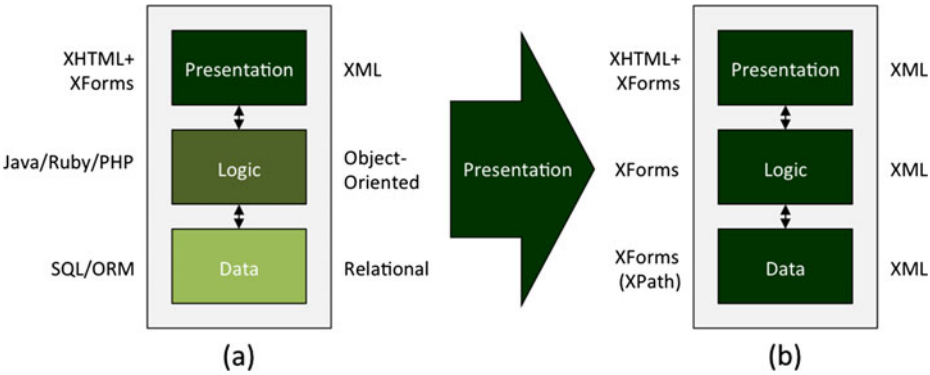

Figure 2 (**a**) The conventional three-tier Web Application architecture using XForms and its (**b**) presentation-centric architectural expansion [29]

common server-side functionality relates to data management, a seamless integration of a standardized query language with the XForms markup language is also important. However, enterprise-level Web Applications are beyond our scope, since we focus on end-user developers, who do not require complex application logic.

Kaufmann and Kossmann [22] listed general requirements for Web Applications that cover all three tiers of a Web Application; including communication requirements. From their list, only four requirements fall within the scope of this paper (cf. R1.1-4 below). In addition, we include three additional general requirements (cf. R1.5-7).

R1.1 Persistence and database: A uniform API for connecting to different types of data sources must be provided. In addition, a standardized, declarative query language applicable across all data sources viewable as XML must be supported.
R1.2 Error handling: A method for notifying the client about errors occurred while processing a requested server-side command must be provided.
R1.3 Session management and security: Managing sessions between a client and a server must be supported regardless of the browser used or its settings. In addition, documents sent to the client must neither expose nor allow unauthorized altering of sensitive information.
R1.4 Modules to facilitate recurring tasks: A method to facilitate modularity and reuse of ready-made components (e.g., user interface parts and queries) in Web Applications must be supported.
R1.5 State maintenance: A method to maintain the state in Web Applications—especially, a mechanism for passing state information (e.g., in the XML format) between documents—must be supported.
R1.6 Authentication, authorization, and access control: A simple way to authenticate users and to handle common access control tasks must be provided.
R1.7 Real-time communication: A uniform API for communicating with different types of RTC servers must be provided.

Furthermore, we define two specific requirements for the language extensions (cf. R2.1-2).

R2.1 Similar syntax and processing model: The syntax and processing model of a language extension must be similar to its base language.

R2.2 Extensible architecture: The architecture of a language extension must provide a method to define new features (e.g., server-side commands) in the language extension while retaining the same processing model.

We want to empower end-user developers below the skill level 9 to create Web Applications. We pursue a smooth transition from the levels 4 and 5 to the levels 6 and 7. To support this transition, we need an IDE, and thus we also define IDE requirements (cf. R3.1-4).

R3.1 Unified background technology: End-user developers face problems, when they have to learn new or combine different technologies. Thus, the background technology must be unified and based on markup languages. Using fewer technologies—or even one markup language—simplifies development of Web Applications [28].
R3.2 Same background technology: Source code editing includes both application and component source code editing. Common background technology removes additional learning barriers. In addition, end-user developers can use existing components as examples of what can be achieved with the technology [42].
R3.3 Transparent background technology: Component and application architecture must be transparent. Component editing and deployment has to be self-explanatory. Thus, internal libraries and complex architectures are undesirable.
R3.4 Smooth transition: The transition from visual editing to source code editing must be smooth. Level 5 and 6 tools must, for example, assist in code editing, give immediate feedback, have fully functional preview, and highlight the link between the source code and the preview.

6 XForms language extensions

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 [12]: 1) script, 2) new data types and libraries, 3) XPath [3] extension functions, 4) new form controls, 5) XForms Actions, 6) custom events, and 7) new serialization formats.

However, certain XForms extension options do not work in browsers, which have either native or plug-in based XForms support, because they require end-users to update the browser's XForms client (i.e., an XForms processor). 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.

In this section, we describe two XForms extensions: XFormsDB and XFormsRTC. The former addresses the requirements R1.1-6, while the latter focuses on the requirement R1.7; both fulfill the requirements R2.1-2.

6.1 XFormsDB

XFormsDB specifies common server-side and database-related functionalities that turn XForms into a comprehensive Web programming language. It fulfills the requirements R1.1-6, as discussed in the following code examples. On the other hand, XFormsRTC complements XForms' HTTP-based communication with two-way, full-duplex, true RTC

capabilities. Thus, XFormsRTC fulfills the requirement R1.7. Both language extensions support requirements R2.1-2 and are mainly targeted at the level 6 end-user developers and allow them to develop and maintain entire Web Applications on their own.

In the following, we use a simple blog application as an example. The XFormsDB Blog application provides basic functionalities for consuming and publishing content, such as news, thoughts, comments, and experiences. The application (http://testbed.tml.hut.fi/blog/) and its source code are both publicly available (cf. Section 7).

Blog users can read published posts and related comments, leave their own comments, and browse through archives. In addition, administrators can manage published posts and comments. The user interface (cf. Figure 3) looks and feels like any other modern Web Application, i.e., it gives a fast response to user input and remains responsive while submitted requests are being processed on the server side. Next, we provide a high-level description of the XFormsDB server-side language extension used in the application, along with the XFormsRTC language extension, proposed in this paper.

Server-side requests are commands submitted to and securely executed on the server side. They are defined within a new *xformsdb:instance* element, which acts as a wrapper

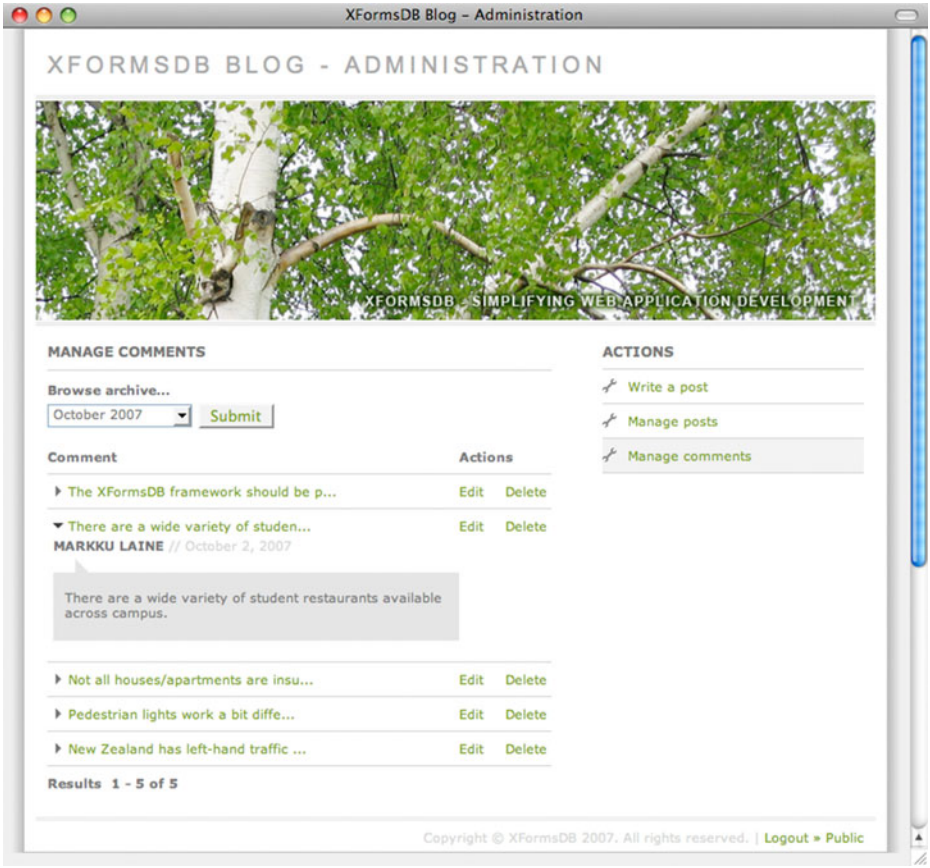

Figure 3 A screenshot image showing the XFormsDB Blog administration user interface

for all server-side requests. This enables adding new commands to the language without requiring any changes to the request-processing model. Currently, the language extension includes definitions for the following server-side commands: 1) maintaining state information, 2) logging users in and out, 3) retrieving information about a currently logged-in user, 4) executing queries against data sources, 5) managing files, and 6) checking the browser support for cookies.

Listing 1 shows an example of a *query* command for retrieving comments of a selected blog post (cf. R1.1). The parameterized query expression is written in XPath [3], whose source code is defined in an external resource (cf. line 3).

```
1:  <xformsdb:instance id="select-and-update-comments-reqinstance">
2:    <xformsdb:query datasrc="exist-db" doc="blog.xml">
3:      <xformsdb:expression resource="../xpath/select_and_update_comments.xpath" />
4:      <xformsdb:xmlns prefix="xformsdb" uri="http://www.tml.tkk.fi/2007/xformsdb" />
5:      <xformsdb:var name="postid" />
6:    </xformsdb:query>
7:  </xformsdb:instance>
```

Listing 1 Definition of a *query* command

The new *xformsdb:submission* element submits server-side requests. As in standard XForms submissions, server-side requests can also be submitted multiple times and at any point in a form's lifetime (cf. R1.5). Listing 2 demonstrates the submission of the aforementioned *query* command. The standard XForms *send* action triggers the submission.

```
1:   <xformsdb:submission id="select-comments-sub" replace="instance"
2:     instance="comments-instance" requestinstance="select-and-update-comments-reqinstance"
3:     expressiontype="select">
4:     <xforms:action ev:event="xforms-submit">
5:       …
6:     </xforms:action>
7:     <xforms:action ev:event="xforms-submit-done">
8:       …
9:     </xforms:action>
10:    <xforms:action ev:event="xformsdb-request-error">
11:      …
12:   </xforms:action>
13: </xformsdb:submission>
```

Listing 2 Definition of a *query* command submission

XForms includes events (e.g., *xforms-ready* and *xforms-submit-done*), which standard event handlers (XForms Actions) catch using XML Events. Custom events are also possible. XFormsDB includes a new *xformsdb-request-error* event, which indicates a failure in a server-side request submission and/or execution process (cf. R1.2). Listing 3 shows the actions to be done in case a server-side error occurs while executing the aforementioned *query* command submission.

```
1:  <xforms:action ev:event="xformsdb-request-error">
2:    <xforms:toggle case="off-progress-animation" />
3:    <xforms:toggle case="select-comments-sub-error-message" />
4:  </xforms:action>
```

Listing 3 The *xformsdb-request-error* event represents server-side errors

Standard XForms lacks a secure mechanism for controlling user access to certain portions of a document. We have extended XForms to include a role-based authorization system (cf. R1.3 and R1.6). The system contains:

1. A new element, *xformsdb:secview*, to control user access within a document.
2. Server-side requests for *logging users in* and *out* as well as *retrieving information about a currently logged-in user*.
3. A realm XML document representing a "database" of usernames, passwords, and roles assigned to those users.

Listing 4 shows how users with roles other than *admin* are redirected to the login Web page (cf. lines 1–5), whereas *admin* users can access the contents between the lines 6–8.

```
1:  <xformsdb:secview>
2:    <xforms:model>
3:      <xforms:load resource="../login.xformsdb" ev:event="xforms-ready" />
4:    </xforms:model>
5:  </xformsdb:secview>
6:  <xformsdb:secview roles="admin">
7:    …
8:  </xformsdb:secview>
```

Listing 4 The *xformsdb:secview* element implements a role-based authorization

The new *xformsdb:include* element provides a recursive inclusion mechanism, which facilitates modularity (cf. R1.4). This element allows the construction of large XML documents from several well-formed XML documents. Compared to XInclude [35], the *xformsdb:include* element is simpler. In addition, its processing model is in line with the other new XFormsDB elements. In the demo application, the *xformsdb:include element* could be used to include common metadata information within the head section on all Web pages (cf. Listing 5).

```
1:  <xformsdb:include resource="../xinc/meta.xinc" />
```

Listing 5 The *xformsdb:include* element supports code reuse

6.2 XFormsRTC

In standard XForms, the communication between a client and a server is typically asynchronous over HTTP(S). Since HTTP is a request-response protocol, the client always needs to initiate the communication with the server. However, there are different ways to emulate server push over HTTP. The simplest way is to *poll* the server at constant time intervals, but this generates lots of unnecessary network traffic. More efficient approach is to use so-called *long-polling* (a common implementation of *Comet* [48]). Long-polling delays the completion of an HTTP response until either next update becomes available or the connection times out.

The left-hand side column of Listing 6 shows how two-way communication—including support for server push updates through long-polling—can be realized in Web Applications using standard XForms. First, two separate *xforms:submission* elements need to be defined, one for each direction of communication. Sending data to the server

happens normally using the *xforms:send* element, which initiates the submission process.

XForms

Constructors

new

```
<xforms:submission id="receive-sub"
  mode="asynchronous"
  method="get"
  resource="http://www.example.com/receivehandler"
  serialization="none"
  replace="instance"
  instance="receive-data-instance">
  <!-- Event handlers -->
  ...
</xforms:submission>

<xforms:submission id="send-sub"
  mode="asynchronous"
  method="post"
  resource="http://www.example.com/sendhandler"
  serialization="application/xml"
  ref="instance( 'send-data-instance' )"
  replace="none">
  <!-- Event handlers -->
  ...
</xforms:submission>
```

Event handlers

onconnect

–

ondisconnect

–

ondata

```
<xforms:action ev:event="xforms-submit-done">
  ...
  <!-- Restart long-polling -->
  <xforms:send submission="receive-sub" />
</xforms:action>
```

onerror

```
<xforms:action ev:event="xforms-submit-error">
  ...
</xforms:action>
```

Methods

connect

–

disconnect

–

send

```
<xforms:send submission="send-sub" ev:event="..." />
```

XFormsRTC

Constructors

new

```
<xformsrtc:connection id="rtc-conn"
  resource="ws://www.example.com"
  serialization="application/xml"
  ref="instance( 'send-data-instance' )"
  replace="instance"
  instance="receive-data-instance">
  <!-- Event handlers -->
  ...
</xformsrtc:connection>
```

Event handlers

onconnect

```
<xforms:action ev:event="xformsrtc-connection-connect">
  ...
</xforms:action>
```

ondisconnect

```
<xforms:action ev:event="xformsrtc-connection-disconnect">
  ...
</xforms:action>
```

ondata

```
<xforms:action ev:event="xformsrtc-connection-data">
  ...
</xforms:action>
```

onerror

```
<xforms:action ev:event="xformsrtc-connection-error">
  ...
</xforms:action>
```

Methods

connect

```
<xformsrtc:connect connection="rtc-conn" ev:event="..." />
```

disconnect

```
<xformsrtc:disconnect connection="rtc-conn" ev:event="..." />
```

send

```
<xforms:send xformsrtc:connection="rtc-conn" ev:event="..." />
```

Listing 6 A comparison between standard XForms (left) and the XFormsRTC language extension proposal (right) for two-way communication. XForms can only emulate RTC using two different connections and a polling-based technique, whereas XFormsRTC provides two-way, full-duplex, true RTC capabilities over a single connection

Reception of server push updates requires the initiation of another submission process; identified by the id *receive-sub*. Whenever new data becomes available from the server or the connection is timed out, the *xforms-submit-done* event is dispatched and the event (i.e., received data) gets handled within the appropriate event handler. At the end of the event handler, the long-polling submission process is restarted to continue receiving further updates from the server. As the example shows, realizing two-way communication using standard

XForms is non-optimal. It requires two separate submissions and re-purposes HTTP to emulate server push. Also, the syntax is complex and unintuitive to use. Therefore, an alternative approach—e.g., similar to WebSocket [14, 19] but with a declarative API—is needed to add easy to use, two-way, full-duplex, true RTC capabilities to XForms.

The right-hand side column of Listing 6 presents our language extension proposal called XFormsRTC. XFormsRTC complements XForms' HTTP-based communication with two-way, full-duplex, true RTC capabilities. It uses a single *xformsrtc:connection* element to define a long-lived, two-way, full-duplex connection between a client and a server. The connection can be opened and closed (cf. the *connect* and *disconnect* methods) multiple times and at any point in a form's lifetime. Otherwise, the syntax and functionality of the element is essentially similar to the standard *xforms:submission* element, which makes it simple and intuitive for end-user developers to adopt. XFormsRTC also introduces four new events: 1) *xformsrtc-connection-connect*, 2) *xformsrtc-connection-disconnect*, 3) *xformsrtc-connection-data*, and 4) *xformsrtc-connection-error*. They are dispatched to indicate changes in connection state or upon receiving server push updates.

XFormsRTC supports different types of RTC servers including Comet and WebSocket. The URI scheme of the *xformsrtc:connection* element's *resource* attribute defines the communication protocol. Thus, an XFormsRTC client can send and receive data to/from an RTC server over HTTP or WebSocket. Furthermore, it is possible to use special application-level protocols, such as XMPP [49–51]; defined by the optional *subprotocol* attribute. Indeed, Pohja [45] proposes that HTTP communication should be complemented with XMPP in order to meet the communication requirements of modern Web Applications. This, of course, requires that the RTC server supports either an HTTP binding for XMPP (BOSH) [43] or a WebSocket binding for XMPP [53], which is still in draft form.

7 XFormsDB implementation

The XFormsDB Blog application presented above was developed using a framework called XFormsDB [27, 29]. The XFormsDB framework is an open source project (http://mediatech.aalto.fi/publications/webservices/xformsdb/). The framework is implemented in pure Java, and includes an XFormsDB processor supporting the above-described XFormsDB server-side language extension.

7.1 XFormsDB framework

The XFormsDB framework is a generic platform for developing and hosting Web Applications based on the XForms markup language and our server-side extensions introduced in the previous section. The framework uses a set of third-party software and libraries, such as 1) *Apache Tomcat* (http://tomcat.apache.org/) HTTP Web server, 2) *eXist-db* (http://exist-db.org/) native XML database (NXD) [38], and 3) *Orbeon Forms* (http://www.orbeon.com/) Ajax-based server-side XForms processor.

Figure 4 depicts the high-level architecture of the XFormsDB framework. At the logic tier, a generic software component called *XFormsDB Processor* replaces the custom server-side software used in conventional XForms-based Web Applications. Thus, all application development happens on the client side using *Extended XHTML+XForms Documents*. The server also includes an Ajax-based *XForms Processor* (i.e., Orbeon Forms), which transforms requested documents into cross-browser (X)HTML+CSS+JavaScript or plain (X)HTML+CSS, depending on the configuration. The

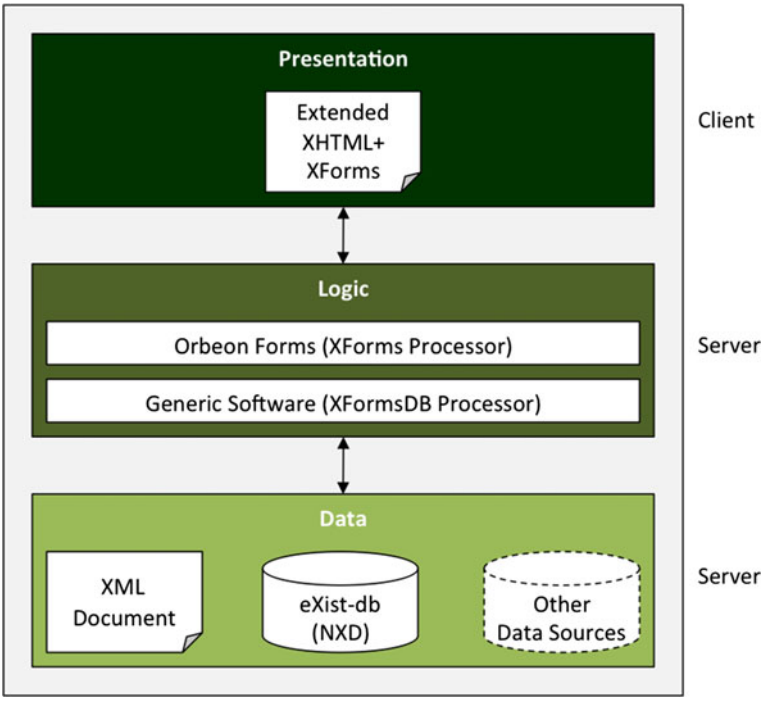

Figure 4 The high-level architecture of the XFormsDB framework

communication between the client and the server happens asynchronously over HTTP(S). Thus, normal browsers can be used as runtime environments. Currently, the framework supports only XML-based data sources, i.e., *XML documents* and *eXist-db*. However, support for *Other Data Sources* (e.g., relational databases) can be easily added with suitable middleware, such as *DataDirect XQuery* (http://www.progress.com/products/data-integration-suite/xquery/xquery-product-architecture/).

7.2 XFormsDB processor

The XFormsDB processor is a generic software component that supports our XFormsDB server-side language extension. The processor's responsibilities include: 1) handling requests and writing responses, 2) transforming extended XHTML+XForms documents, 3) managing sessions, 4) performing synchronized updates, and 5) providing integration services to heterogeneous data sources. Each of these responsibilities is carried out by a separate component.

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 redirects the request to an appropriate request handler. Then, XFormsDB Transformer processes the document as follows:

1. Parse the document and identify server-side extension elements.
2. Incorporate all external documents into the main document (cf. *xformsdb:include*).
3. Filter out those parts to which the user does not have access rights (cf. *xformsdb:secview*).

4. Identify and collect information about other relevant elements (e.g., *xformsdb:instance*, *xformsdb:query*, and *xformsdb:submission*).
5. Store the information found in the previous step into the session (*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 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 the document into a format viewable by the requesting client.

Also asynchronous form submissions over HTTP(S) go through the front controller (*XFormsDB Servlet*). The front controller extracts relevant request information and forwards the request to an appropriate request handler based on the submitted command. When a *query* command submission occurs, the original query expression is fetched from the session (*XFormsDB Managers*) using the opaque reference ID and executed against the underlying data source (*XML Document* and *eXist-db Adapters*). For update *query* command submissions, the XFormsDB processor provides a simple and elegant XPath-based solution for performing synchronized updates, which the *3DM XML 3-Way Merger* component [30] performs. Finally, a response XML is composed and returned to the client.

7.3 Extensibility and limitations

The XFormsDB framework supports extensibility on all three Web Application tiers. On the presentation tier, JavaScript enhances animations, interactivity, and client-side application logic. In addition, eXist-db's XQuery extension functions (http://exist-db.org/exist/apps/fundocs/browse.html?extensions=true) augment the server-side application logic. Also, more expressive XQuery [4] overcomes XPath's limitations, such as lack of grouping, sorting, and cross-document joins. The XFormsDB framework also provides an elegant way to define new server-side requests to the language extension. For validating the structures and data types of transmitted XML documents, the same XML Schema [13] can be used both on the client and the server. These extension methods make XFormsDB fully compatible with the XRX (XForms/REST/XQuery) architecture [37], and thus makes it a viable option over the conventional three-tier Web Applications architecture [40].

Unfortunately, each of the aforementioned methods require developers to a learn a new technology before use. In addition, the XFormsDB framework assumes knowledge of W3C-standardized XForms and the syntax of our extension, which may be a barrier for lower-level end-user developers. Therefore, to widen the group of potential end-user developers utilizing the platform, an end-user dedicated IDE with special assistance features (e.g., syntax highlight and component-based development) is required.

8 XFormsDB integrated development environment

The XFormsDB IDE (XIDE) (http://mediatech.aalto.fi/publications/webservices/xide/) is a visual end-user tool that assists lower-level end-user developers in Web Applications development [31]. XIDE supports multiple Web Application development levels (cf. Figure 1). It

gently leads level 4 end-user developers to the upper levels 5–7 without introducing major learning barriers.

XIDE has four development views: 1) getting started, 2) list of applications, 3) application management, and 4) page editing. Each view contains information and functionality related to the activity at hand. Extending the approach of [34], XIDE provides several levels of modification, such as customization of components, visual manipulation, page source code modification, and component source code modification. In addition, end-user developers can leverage XFormsDB skills they have gained, when they try more complex tasks.

XIDE combines three approaches to achieve this goal, as depicted in Figure 5. First, it helps end-user developers to leverage their existing level 1–4 skills. For example, end-user developers can create Web Applications using predefined, customizable components (level 1). Also, components and pages have WYSIWYG-like representations (level 2), which reveal their contents. In addition, visual editor (level 3) allows end-user developers visually add components to the page and connect them. XIDE supports direct manipulation of components, i.e., drag-n-drop for adding, managing, and deleting components. Finally, end-user developers can edit the source code of each component, written in the markup-based language (level 4).

Second, XIDE provides an extensive source code editing functionality. The source code is written using markup languages, and thus the editing activity belongs to the level 4. The language is unified, so it allows building Web Applications, including server-side database

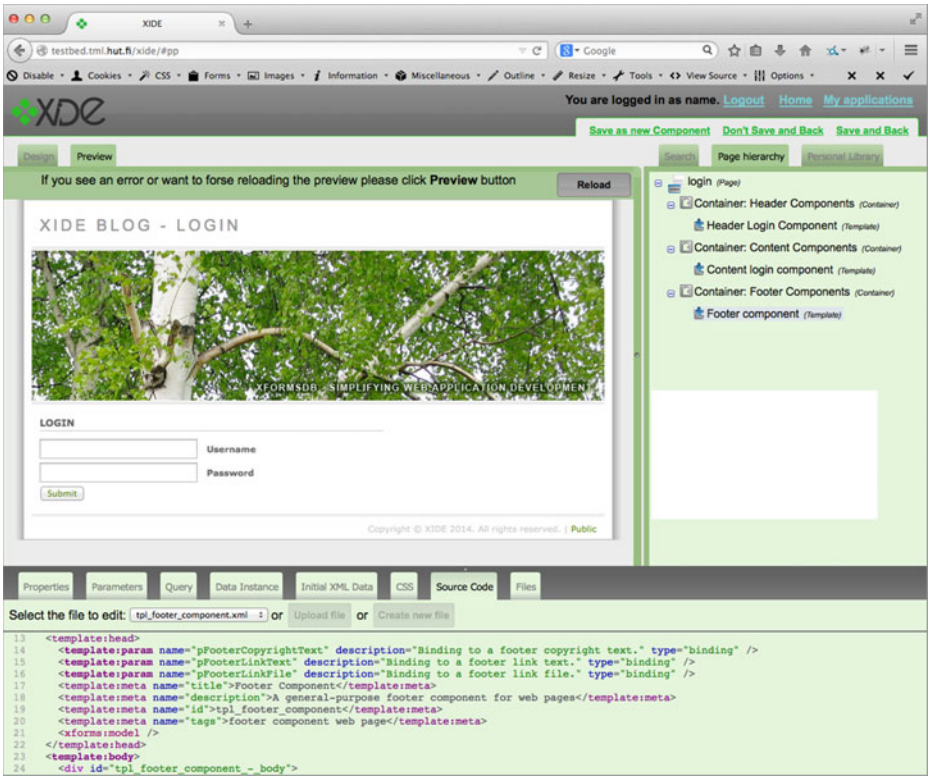

Figure 5 Developing the blog application with XIDE. The preview of the login page is shown in the top-left corner, the components used for building the page are listed on the right, and the source code of the selected footer component is displayed at the bottom

functionality. Hence, end-user developers can achieve the tasks from the level 5. Moreover, end-user developers can incrementally learn the technology and succeed in more advanced code editing (levels 6 and 7). XIDE provides wizards and intelligent automatic source code generation for configuring technical issues that can otherwise be complicated for end-user developers. The built-in text editor supports advanced highlighting and error checking, including XFormsDB syntax and logical structure. The syntax highlighting focuses end-user developers' attention to the most important parts of the source code, such as parameters.

Third, XIDE strives to reveal the connection between visual and source code editing, which is a challenging issue for end-user developers [24]. XIDE uses multiple coordinative windows to show one concept from different perspectives or different degrees of abstraction. This allows end-user developers, for example, to discover the link between a component source code and its visual output. XIDE also supports design at runtime: changes made to the page or component source code instantly appear in the design or preview representation.

Developing the blog application with XIDE To demonstrate XIDE's capabilities, we developed the blog application introduced in Section 6.1 using XIDE (cf. Figure 5). To support visual component-based development, we created (levels 6 and 7) three reusable components: header, login form, and footer. For each component, it is possible to define own queries, data instances, database data, CSS, and resources (e.g., images). In addition, components can be parameterized to increase their reusability. Moreover, XIDE allows to edit and customize existing components, which can then be saved as new components (levels 4 and 5). The content of the components was defined in XHTML, XForms, and XFormsDB.

Then, we developed three pages of the application as end-users would do. For example, the login page (shown in the preview area), uses all three of the aforementioned components to construct the Web page. The components were dragged (levels 1–3) into dedicated component containers, and as a result, XIDE generated bindings between the components and Web pages. The other two pages were developed using both existing components (levels 4 and 5) and manual XFormsDB coding (levels 5–7). Finally, the blog application was published online on the XIDE platform: http://testbed.tml.hut.fi/fakeuser/xideblogdemo/.

9 Evaluation

In this section, we evaluate the XFormsDB framework and the XIDE visual tool from three different perspectives. First, we evaluate XFormsDB as a software product. Then, we analyze different XFormsDB-based Web Applications. Finally, we report the results of XIDE user tests.

9.1 Software product quality

Software product quality characteristics are part of the ISO/IEC 25010:2011 standard [21]. However, the standard is not specifically designed to evaluate Web Application frameworks, and thus we include two additional characteristics: time-to-market [41] and support. The latter includes documentation quality, development community activity, and development tool availability.

Functional Suitability: XFormsDB's functionality covers only the most common server-side and database-related tasks, because the framework targets end-user developers whose

main focus is not on the application's business logic. Thus, it should be used to develop small and medium-sized Web Applications instead of enterprise-level applications containing complex business logic (cf. Section 9.2).
Reliability: XFormsDB is an academic project. However, its reliability has been tested on numerous Web Applications (cf. Section 9.2).
Security: XFormsDB provides built-in and easy-to-use security capabilities for authentication, authorization, and access control (cf. Section 6.1).
Compatibility: XFormsDB can consume data from third-party systems using, for example, Representational State Transfer (REST) APIs. In addition, XFormsDB can expose its application data through eXist-db's RESTful API.
Operability: For an end-user with little technical knowledge, XFormsDB is easy to learn (cf. Section 9.3), because it is based on declarative languages with limited functionality. The adoption rate of both XForms and XQuery, however, is still quite low, despite being W3C recommendations.
Performance efficiency: XFormsDB response times are reasonable but not highly optimized [27]. Moving XFormsDB-based Web Applications to the cloud for a better scalability requires further study.
Maintainability: XFormsDB relies heavily on declarative W3C standards, which increases code modularity and reusability. In addition, XFormsDB is an open-source project, so it can be easily modified and extended.
Portability: The XFormsDB development environment comes with executable scripts and a bundled Web server, which makes it easy to install and uninstall.
Time-to-Market: XFormsDB allows rapid prototyping but requires additional learning up front.
Support: XFormsDB has a visual development tool XIDE. XFormsDB's documentation is good, but it is mostly restricted to scientific publications.

9.2 Applications

The XFormsDB Blog application is analyzed in detail in [27], which contains also an analysis of a Personal Information Management (PIM) application. In addition, [29] presents an additional XFormsDB-based application: Project Management.

Åkerberg [1] did a more comprehensive evaluation of XFormsDB application development. He developed a mobile NFC-based ticketing application without any previous knowledge on XForms or XFormsDB. The application has three main views: 1) home screen, 2) route screen, and 3) payment screen. The home screen shows the most often needed travel card information. The route screen displays the current traveler's location as well as the nearest public transportation stations and stops. The payment screen allows loading travel value or period to the customer's travel card.

At the time of the development of the mobile ticketing application, XIDE was not ready, and thus the application was implemented directly using the XFormsDB framework. The biggest problems arose from the lack of clear instructions and examples of how certain functionality should be implemented using XForms. Otherwise, XFormsDB proved to be an efficient platform for the mobile ticketing application. In addition to the internal XFormsDB database, the application used several external APIs. The integration of HTTP-based APIs was simple, at least when the returned data was in the XML format.

9.3 XIDE evaluation

We conducted a qualitative evaluation of XIDE with nine participants (5 males and 4 females) [32]. They were 22–31 years old (average age 25 years). The aim of the study was to investigate whether level 4 end-users could perform level 6 and 7 tasks without facing any major learning barriers. We recruited the participants from Economics and Computer Science students. We only accepted level 4 participants: they were active Internet users, had tried to create a simple Web page using a visual editor, and tried to use some XML-based markup language. None of the participants studied Web development as a major. None of them had ever used XIDE, XFormsDB, or XForms.

Each person participated in the study individually and each evaluation took approximately 1.5–2 hours. First, the participants were interviewed individually about their Web Application development experience. After that, each participant was given a brief introduction to the XIDE tool. Then, each participant was given a set of tasks, ranging from simple visual manipulations to direct code editing. An evaluator observed how the participants executed the tasks and discussed the motivation and decisions made with each participant. At the end, we performed a closing interview.

Each participant performed ten tasks, which were designed to resemble the transition from visual editing (level 3) to XFormsDB programming (level 7). First, they used the visual editor for simple modifications (level 3). Then, they executed the same tasks manually to the source code (level 4). After that, the participants were asked to modify the page source code (level 5). Next, they had to do first minor (level 5) and then major modifications to the components (level 6–7). Finally, the participants had to contribute new components (level 6–7).

We used a think-aloud method to gather data about participants' performance. Participants talked about the XIDE interface. They described and explained their actions. An interviewer was allowed to ask qualifying questions. This allowed us to better understand how participants perceived the XIDE user interface and tasks. However, performance times of individual tasks did not provide significant results.

All participants were able to perform all the given tasks. In general, the participants said they found the XIDE approach promising and would use such tool. They found it useful and flexible, because it supported different levels of modification. In overall, all nine participants were able to smoothly transit from the level 4 to the levels 6–7 without facing any major learning barriers.

The participants widely employed the copy-paste approach. XIDE provided neither vocabulary of available XFormsDB or HTML tags that could have been used nor code completion. Thus, the only way a novice user could contribute new code was to find the piece of code that had the same functionality, copy it, and modify it according to the task. In such tasks, immediate response from the system was highly appreciated. The participants could see what had changed in the page after his/her modifications to the source code. This also helped them to understand whether it was the right place to do the modifications.

The source code highlighting was an extremely useful feature. Less important parts—such as the data model or head part—were displayed in grayscale. Most participants started looking into them at the very end, and thus concentrated first on more valuable parts. The participants' attention was focused on the code part, which—most likely—contained the needed information. On the other hand, the rest of the code was still visible, so the participants could check it if needed. All participants named this feature among the most usable features in XIDE.

The XIDE evaluation provided useful information, but had also some validity limitations. First, the participants progressed from task to task and from level to level. In a

real-life situation, users might have problems in decomposing a task into smaller steps. Second, participants were selected from a narrow group of university students who were about the same age.

10 Conclusions

This paper contains three new contributions: 1) extended nine-level classification of Web Application development, 2) requirements for real-time communication capabilities in Web Applications, and 3) new XFormsRTC language extension for client–server communication in XForms-based Web Applications.

First, we investigated how the complexity of Web Application development increases when proceeding from traditional end-user development towards conventional three-tier Web Applications development. We defined nine Web Application development levels. The first three levels are within the reach of most end-user developers: level 1) customizing components, level 2) WYSIWYG editing, and level 3) visual editing. More advanced end-user developers master the level 4) markup authoring, and even the level 5) snippet programming. However, most end-user developers are reluctant to become real programmers. We divided professional Web development into four levels: level 6) single language programming, level 7) unified language programming, level 8) multiple language programming, and level 9) multiple language and paradigm programming. Using this analysis as a basis, we strived to reassess how to expand the scope of Web Applications mid-level end-user developers can create without facing major learning challenges.

We based our approach on the level 7) unified language programming. In our proposal, the presentation tier is expanded to cover all three tiers of a Web Application. This allows end-user developers not only to leverage their existing skills in user interface development, but also to implement entire Web Applications using a single declarative language and data model. In the proposed XFormsDB framework, all application development is done on the client side. We believe that this helps especially Web designers—usually mid-level end-user developers—to become advanced Web Application developers. The framework is based on the XForms markup language and our proposed XFormsDB server-side language extension. We implemented the XFormsDB framework based on the derived requirements, and argued that it could simplify both the development and maintenance of small and medium-sized Web Applications.

In addition, we proposed an improvement for the client–server communication in XForms-based Web Applications. Our proposal—the XFormsRTC language extension—complements XForms' HTTP-based communication with two-way, full-duplex, true RTC capabilities. Using the XFormsRTC uniform API, it is possible to communicate with different types of RTC servers in a fully declarative manner.

To assist lower-level end-user developers in the development of XFormsDB-based applications, we described and implemented the XIDE editor: a visual Web Application development tool supporting all the above-mentioned Web programming activities. XIDE aims to gently lead end-user developers from the level 4) markup authoring to the level 5) snippet programming, and even further to the levels 6) single language programming and 7) unified language programming. The XFormsDB framework and the XIDE visual authoring tool (together with a number of examples) are available under the MIT license.

In the future, we plan to improve XFormsDB data access. Moreover, we will introduce a unified querying approach that allows XForms clients to consume heterogeneous

data on the Web (e.g., JSON data via Web APIs) in a developer friendly and efficient manner.

References

1. Åkerberg, M.: Evaluation of XFormsDB-Based Application Development—A Case Study. M.Sc. Thesis. Aalto Univ., Espoo, Finland (2010)
2. Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web Services: Concepts, Architectures and Applications (1st ed.). Springer (2004). ISBN: 978-3-540-44008-6
3. Berglund, A., Boag, S., Chamberlin, D., Fernández, M.F., Kay, M., Robie, J., Siméon, J. (eds.): XML Path Language (XPath) 2.0 (Second Edition). W3C Recomm. http://www.w3.org/TR/xpath20/ (2010). Accessed 7 Feb. 2014
4. Boag, S., Chamberlin, D., Fernández, M.F., Florescu, D., Robie, J. Siméon, J. (eds.): XQuery 1.0: An XML Query Language (Second Edition). W3C Recomm. http://www.w3.org/TR/xquery/ (2010). Accessed 7 Feb. 2014
5. Boyer, J.M. (ed.): XForms 1.1. W3C Recomm. http://www.w3.org/TR/xforms/ (2009). Accessed 7 Feb. 2014
6. Boyer, J.M., Klotz, Jr., L.L., Pemberton, S., Van den Bleeken, N. (eds.): XForms 2.0. W3C Working Draft. http://www.w3.org/TR/xforms20/ (2012). Accessed 7 Feb. 2014
7. Bozzon, A., Comai, S., Fraternali, P., Toffetti Carughi, G.: Conceptual Modeling and Code Generation for Rich Internet Applications. In: Proc. 6th International Conference on Web Engineering (ICWE), pp. 353–360 (2006). doi:10.1145/1145581.1145649
8. Brandt, J., Guo, P.J., Lewenstein, J., Klemmer, S.: Opportunistic Programming: How Rapid Ideation and Prototyping Occur in Practice. In: Proc. 4th Int. Workshop End-User Softw. Eng. (WEUSE'08), pp. 1–5 (2008). doi:10.1145/1370847.1370848
9. Cappiello, C., Matera, M., Picozzi, M., Sprega, G., Barbagallo, D., Francalanci, C.: DashMash: A Mashup Environment for End User Development. In: Auer, S., Díaz, O., Papadopoulos, G.A. (eds.) Web Engineering (ICWE'11), Lect. Notes Comput. Sci. Springer, 6757, pp. 152–166 (2011). doi:10.1007/978-3-642-22233-7_11
10. Casteleyn, S., Garrigós, I., Mazón, J.-N.: Ten Years of Rich Internet Applications: A Systematic Mapping Study, and Beyond. ACM Trans. Web **8**(3), 18:1–46 (2014). doi:10.1145/2626369
11. Costabile, M.F., Mussio, P., Parasiliti Provenza, L., Piccinno, A.: End Users as Unwitting Software Developers. In: Proc. 4th Int. Workshop End User Softw. Eng. (WEUSE'08), pp. 6–10 (2008). doi:10.1145/1370847.1370849
12. Dubinko, M.: XForms Essentials (1st ed.). O'Reilly Media (2003). ISBN: 978-0-596-00369-2
13. Fallside, D.C., Walmsley, P. (eds.): XML Schema Part 0: Primer Second Edition. W3C Recomm. http://www.w3.org/TR/xmlschema-0/ (2004). Accessed 7 Feb. 2014
14. Fette, I., Melnikov, A. (eds.): The WebSocket Protocol. IETF RFC 6455 (Proposed Stand.), https://tools.ietf.org/html/rfc6455 (2011). Accessed 7 Feb. 2014
15. Fraternali, P., Comai, S., Bozzon, A., Toffetti Carughi, G:. Engineering Rich Internet Applications with a Model-Driven Approach. ACM Trans. Web **4**(2), 7:1–47 (2010). doi:10.1145/1734200.1734204
16. Garrett, J.J.: Ajax: A New Approach to Web Applications, http://www.adaptivepath.com/ideas/ajax-new-approach-Web-applications (2005). Accessed 7 Feb. 2014
17. Heinrich, M., Gaedke, M.: Data Binding for Standard-Based Web Applications. In: Proc. 27th Annual ACM Symp. Appl. Comput. (SAC'12), pp. 652–657 (2012). doi:10.1145/2245276.2245402
18. Hickson, I. (ed.): HTML5: A Vocabulary and Associated APIs for HTML and XHTML. W3C Recomm. http://www.w3.org/TR/html5/ (2014). Accessed 16 Mar. 2015
19. Hickson, I. (ed.): The WebSocket API. W3C Candidate Recomm. http://www.w3.org/TR/Websockets/ (2012). Accessed 7 Feb. 2014
20. Honkala, M., Vuorimaa, P.: XForms in X-Smiles. World. Wide. Web **4**(3), 151–166 (2001) doi:10.1023/A:1013853416747
21. ISO/IEC 25010:2011: Systems and Software Engineering—Systems and Software Quality Requirements and Evaluation (SQuaRE)—System and Software Quality Models, International Organization for Standardization/International Electrotechnical Commission (2011)
22. Kaufmann, M., Kossmann, D.: Developing an Enterprise Web Application in XQuery. Tech. Rep. ETH Zürich. http://download.28msec.com/sausalito/technical_reading/enterprise_webapps.pdf (2008). Accessed 7 Feb. 2014

23. Ko, A.J., Abraham, R., Beckwith, L., Blackwell, A., Burnett, M., Erwig, M., Scaffidi, C., Lawrance, J., Lieberman, H., Myers, B., Rosson, M.B., Rothermel, G., Shaw, M., Wiedenbeck, S.: The State of the Art in End-User Software Engineering. ACM Comput. Surv., **43**(3), 21:1–44 (2011). doi:10.1145/1922649.1922658
24. Ko, A.J., Myers, B.A., Aung, H.H.: Six Learning Barriers in End-User Programming Systems. In: Proc. Symp. Visual Lang. and Hum. Centric Comput., pp. 199–206 (2004). doi:10.1109/VLHCC.2004.47
25. Krasner, G.E., Pope, S.T.: A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80. J. Object-Oriented. Program. **1**(3), 26–49 (1988)
26. Kuuskeri, J., Mikkonen, T.: Partitioning Web Applications Between the Server and the Client. J. Web. Eng. 9(3), 207–226 (2010)
27. Laine, M.: XFormsDB—An XForms-Based Framework for Simplifying Web Application Development. M.Sc. Thesis. Aalto Univ., Espoo, Finland. http://urn.fi/URN:NBN:fi:aalto-201203131406 (2010). Accessed 16 Mar. 2015
28. Laine, M., Shestakov, D., Litvinova, E., Vuorimaa, P.: Toward Unified Web Application Development. **13**(5), 30–36 (2011). doi:10.1109/MITP.2011.55
29. Laine, M., Shestakov, D., Vuorimaa, P.: XFormsDB: An Extensible Web Application Framework Built upon Declarative W3C Standards. ACM SIGAPP Appl. Comput. Rev. **12**(3), 37–50 (2012). doi:10.1145/2387358.2387361
30. Lindholm, T.: A Three-Way Merge for XML Documents. In: Proc. 2004 ACM Symp. Doc. Eng. (DocEng'04), pp. 1–10 (2004). doi:10.1145/1030397.1030399
31. Litvinova, E.: XIDE—A Visual Tool for End User Development of Web Applications. M.Sc. Thesis. Univ. Eastern Finland, Joensuu, Finland (2010)
32. Litvinova, E., Laine, M., Vuorimaa, P.: XIDE: Expanding End-User Web Development. In: Proc. 8th Int. Conf. Web Inf. Syst. and Tech. (WEBIST'12), pp. 123–128 (2012). doi:10.5220/0003934201230128
33. Lizcano, D., Alonso, F., Soriano, J., López, G.: A New End-User Composition Model to Empower Knowledge Workers to Develop Rich Internet Applications. J. Web. Eng. **10**(3), 197–233 (2011)
34. MacLean, A., Carter, K., Lövstrand, L., Moran, T.: User-Tailorable Systems: Pressing the Issues with Buttons. In: Proc. SIGCHI Conf. Hum. Factors Comput. Syst.: Empowering People (CHI'90), pp. 175–182 (1990). doi:10.1145/97243.97271
35. Marsh, J., Orchard, D., Veillard, D. (eds.): XML Inclusions (XInclude) Version 1.0 (Second Edition). W3C Recomm. http://www.w3.org/TR/xinclude/ (2006). Accessed 7 Feb. 2014
36. Martinez-Ruiz, F., Muñoz Arteaga, J., Vanderdonckt, J., Gonzalez-Calleros, J., Mendoza, R.: A First Draft of a Model-Driven Method for Designing Graphical User Interfaces of Rich Internet Applications. In: 4th Latin American Web Congress (LA-Web), pp. 32–38 (2006). doi:10.1109/LA-WEB.2006.1
37. McCreary, D.: Introducing the XRX Architecture: XForms/REST/XQuery. http://datadictionary.blogspot.fi/2007/12/introducing-xrx-architecture.html (2007). Accessed 7 Feb. 2014
38. Meier, W.: eXist: An Open Source Native XML Database. In: Chaudhri, A.B., Jeckle, M., Rahm, E., Unland, R. (eds.) Web, Web-Services, and Database Systems, Lect. Notes Comput. Sci. Springer, 2593, pp. 169–183 (2003). doi:10.1007/3-540-36560-5_13
39. Mikkonen, T., Taivalsaari, A.: Web Applications – Spaghetti Code for the 21st Century. In: 6th Int. Conf. Softw. Eng. Research, Management and Applications (SERA'08), pp. 319–328 (2008). doi:10.1109/SERA.2008.16
40. Nemeş, C., Podean, M., Rusu, L.: XRX: The Implementation Process under XRX Architecture. In: Proc. 8th Int. Conf. Web Inf. Syst. and Tech. (WEBIST'12), pp. 103–109 (2012). doi:10.5220/0003931101030109
41. Offutt, J.: Quality Attributes of Web Software Applications. IEEE Softw. **19**(2), 25–32 (2012). doi:10.1109/52.991329
42. Park, T. H., Wiedenbeck, S.: First Steps in Coding by Informal Web Developers. In: Proc. Vis. Lang. and Hum.-Centric Comput. (VL/HCC 2010), pp. 79–82 (2010). doi:10.1109/VLHCC.2010.20
43. Paterson, I., Saint-Andre, P., Stout, L., Tilanus, W. (eds.): XEP-0206: XMPP Over BOSH. Draft Standard, XMPP Standards Foundation. http://xmpp.org/extensions/xep-0206.html (2014). Accessed 16 Mar. 2015
44. Pohja, M.: Comparison of Common XML-Based Web User Interface Languages. J. Web. Eng. **9**(2), 95–115 (2010)
45. Pohja, M.: Server Push for Web Applications via Instant Messaging. J. Web. Eng. **9**(3), 227–242 (2010)
46. Repenning, A., Ioannidou, A.: What Makes End-User Development Tick? 13 Design Guidelines. In: Lieberman, H., Paternò, F., Wulf, V. (eds.) End User Development, Hum.-Comput. Interact. Ser. Springer, 9, pp. 51–85 (2006). doi:10.1007/1-4020-5386-X_4
47. Rode, J., Bhardwaj, Y., Pérez-Quiñones, M.A., Rosson, M.B., Howarth, J.: As Easy as "Click": End-User Web Engineering. In: Lowe, D. Gaedke, M., (eds.) Web Engineering, Lect. Notes Comput. Sci. Springer, 3579, pp. 478–488 (2005). doi:10.1007/11531371_61
48. Russell, A.: Comet: Low Latency Data for the Browser. http://infrequently.org/2006/03/comet-low-latency-data-for-the-browser/ (2006). Accessed 7 Feb. 2014

49. Saint-Andre, P.: Streaming XML with Jabber/XMPP. IEEE Internet. Comput. **9**(5), 82–89 (2005). doi:10.1109/MIC.2005.110
50. Saint-Andre, P. (ed.): Extensible Messaging and Presence Protocol (XMPP): Core. IETF RFC 6120 (Proposed Stand.). https://tools.ietf.org/html/rfc6120 (2011). Accessed 7 Feb. 2014
51. Saint-Andre, P. (ed.): Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence. IETF RFC 6121 (Proposed Stand.). https://tools.ietf.org/html/rfc6121 (2011). Accessed 7 Feb. 2014
52. Schmitz, P:. The SMIL 2.0 Timing and Synchronization Model: Using Time in Documents. Tech. Rep. MSR- TR-2001-01. Microsoft Research. http://research.microsoft.com/pubs/69839/tr-2001-01.doc (2001). Accessed 7 Feb. 2014
53. Stout, L., Moffitt, J., Cestari, E. (eds.): An XMPP Sub-protocol for WebSocket. IETF Internet Draft. https://tools.ietf.org/html/draft-moffitt-xmpp-over-websocket-04 (2013). Accessed 16 Mar. 2015
54. Toffetti, G., Comai, S., Preciado, J.C., Linaje, M.: State-of-the Art and Trends in the Systematic Development of Rich Internet Applications. J. Web. Eng. **10**(1), 70–86 (2011)
55. Valverde, F., Pastor, O., Valderas, P., Pelechano, V.: A Model-Driven Engineering Approach for Defining Rich Internet Applications: A Web 2.0 Case Study. In: Handbook of Research on Web 2.0, 3.0 and X.0: Technologies, Business and Social Applications, IGI Global, pp. 40–58 (2010). doi:10.4018/978-1-60566-384-5.ch003
56. Won, M., Stiemerling, O., Wulf, V.: Component-Based Approaches to Tailorable Systems. In: Lieberman, H., Paternò, F., Wulf, V. (eds.) End User Development, Hum.-Comput. Interact. Ser. Springer, 9, pp. 115–141 (2006). doi:10.1007/1-4020-5386-X_6
57. Yang, F., Gupta, N., Gerner, N., Qi, X., Demers, A., Gehrke, J., Shanmugasundaram, J.: A Unified Platform for Data Driven Web Applications with Automatic Client–server Partitioning. In: Proc. 16th Int. Conf. World Wide Web (WWW'06), pp. 341–350 (2007). doi:10.1145/1242572.1242619