Toward Unified Web Application Development

Markku Laine, Denis Shestakov, Evgenia Litvinova, and Petri Vuorimaa,
Aalto University School of Science

Web application development in the context of the conventional three-tier architecture is complex, typically requiring a team of experts. Recent Web application architectures and frameworks simplify the development process, potentially turning tier-specific experts into one-person developer teams.

Traditional Web application development requires a good understanding of heterogeneous systems, programming languages, concepts, and frameworks. For example, in a Web application based on the conventional three-tier architecture,[1] the presentation (user interface), logic (server), and data (data management) tiers are all authored using conceptually different programming languages.[2]

Novel Web application architectures have emerged that aim to reduce this complexity by expanding the concepts used on a single tier to cover all three tiers. In addition to reducing the amount of required knowledge and simplifying the development process, these architectural approaches also allow tier-specific experts to author an entire Web application on their own.

Classification of Programming Languages

Numerous programming languages have been developed over the years. One way of better understanding their similarities and differences is to classify them into a hierarchy of paradigms based on their computation model. Typically, top-level classification comprises two paradigms—imperative (such as C, Java, and JavaScript) and declarative (such as CSS, HTML, and SQL)—which we can further divide into subparadigms, as shown in Figure 1.[3]

The main difference between the two top-level paradigms is that declarative languages specify *what* a program should do (the desired results), rather than *how* to do it (the algorithms needed to achieve the results). This higher abstraction level makes declarative languages easier to reason about and raises their semantic level.

Imperative languages, on the other hand, have more expressive power but are generally harder to use and understand.[4] This classification of top-level paradigms isn't absolute though—some programming languages are more declarative than others, and some even support multiple paradigms.

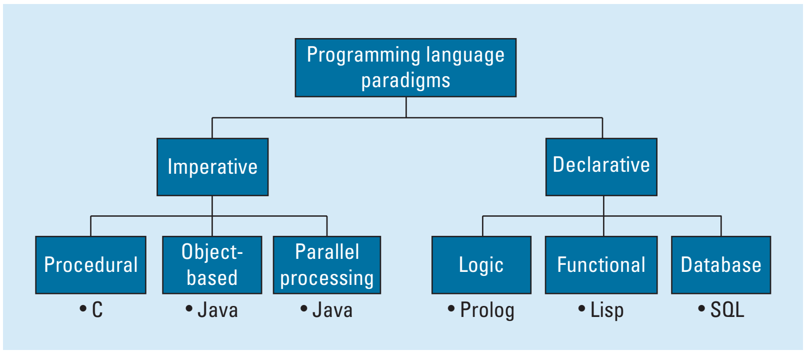

Figure 1. A hierarchy of programming language paradigms (based on Doris Appleby and Julius VandeKopple's work[3]). The top-level classification comprises two paradigms—imperative and declarative.

Another notable difference between imperative and declarative languages is their scope of use. Typically, imperative languages (such as C and Java) are more general-purpose and applicable for solving a variety of problems, whereas declarative languages are more applicable for solving domain-specific problems, such as defining the structure (HTML) or styling (CSS) of a webpage, or managing data (SQL) stored in a relational database. This, of course, makes a certain declarative language a good fit for solving the problems for which it was specifically designed and a poor fit for others because it lacks expressiveness in that domain.

The Evolution of Web Application Architectures

As Figure 2a shows, authors (that is, Web designers, Web developers, and database experts) implement a typical Web application's presentation tier using declarative XHTML, combined with imperative JavaScript to provide additional interactivity. They implement the application logic residing on the server side using an object-oriented imperative language, such as Java, Ruby, or PHP. They carry out communication between these two tiers using declarative formats (for example, XML or JSON—JavaScript Object Notation) and the Ajax technique. Finally, on the undermost tier of the application—that is, the data tier—they use either an object-relational mapping (ORM) library or declarative SQL statements to manage data stored in a relational database.

The problem with this conventional architecture is that it requires different tier-specific experts because of the various programming languages, programming paradigms, and data models on each tier. In addition, the partitioning of a Web application between the client (presentation tier) and the server (logic and data tiers) makes the development process more complex.[5]

One way of simplifying the development process is to expand what is normally used on a single tier to cover all three tiers. An author could also select a single programming language and paradigm (imperative or declarative) from a specific tier to use throughout the entire Web application.

Expanding the Presentation Tier: XFormsDB

A Web designer working on the presentation tier is, as a rule, familiar with declarative XHTML and CSS but not skillful in the server-side aspects of a Web application. Such an expert can benefit from an architectural approach expanding a presentation-centric language with common server-side and database-related functionality. XFormsDB (developed by Markku Laine, http://code.google.com/p/xformsdb) is an example of presentation-centric architectural expansion.[6]

The idea behind the declarative XFormsDB markup language[7] is to naturally extend XForms (see the sidebar) with the most common server-side and database-related functionality (see Figure 2b) so authors no longer have to resort to imperative client-side scripting or server-side programming languages. To manage the data stored in data sources, XFormsDB uses declarative XPath, but it also lets authors use the declarative XQuery to execute more complex queries (see the sidebar). The XFormsDB framework is bundled with the Apache Tomcat Web server, the eXist-db native XML database,[8] and the Orbeon Forms Ajax-based server-side XForms processor, which can transform requested webpages into cross-browser JavaScript and HTML files.

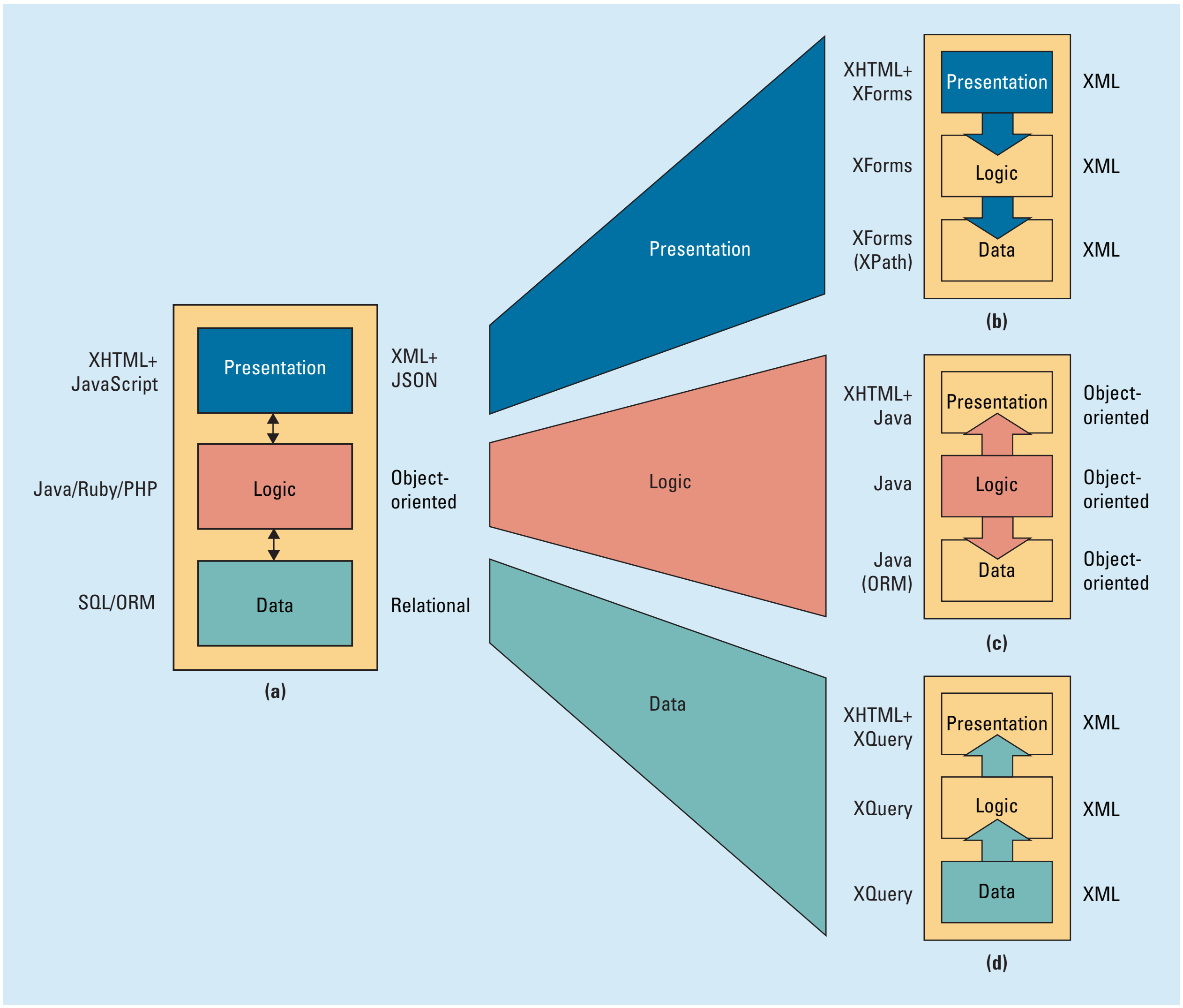

Figure 2. **Presentation, logic, and data-centric approaches for expanding (a) the conventional three-tier Web application architecture. Different Web application frameworks exist for each tier: (b) XFormsDB expands the presentation tier, (c) the Google Web Toolkit is for the logic tier, and (d) Sausalito expands the data tier. For each architecture, technologies appear on the left and data models are on the right.**

Expanding the Logic Tier: The Google Web Toolkit

A Web developer working on the logic tier possesses excellent technical knowledge but doesn't necessarily have the need or motivation to implement a fancy user interface for a Web application. A logic-centric approach based on an imperative, general-purpose programming language is the most attractive option in this case. In 2006, Google introduced its Google Web Toolkit (GWT, http://code.google.com/webtoolkit), which became one of the first modern frameworks for this approach.

GWT aims to expand a general-purpose programming language—namely, object-oriented imperative Java—beyond its intended scope so that it can be used not only to author the server-side application logic but also a Web application user interface (see Figure 2c). (Authors could also use XHTML to create the user interface). When a GWT-based Web application is deployed, the GWT Java-to-JavaScript compiler translates the user interface code written in Java into cross-browser JavaScript and HTML files, thus making it possible to create complex Ajax-style Web applications in Java.

This approach also supports the execution of the same application code on both the client and server side, increasing the code's reusability.[5] Furthermore, using GWT with an ORM

XForms and XQuery

Here we introduce two W3C recommendations for declarative Web application development.

XForms: An XML Form Language

XForms, a W3C recommendation since October 2003, is an XML-based Web forms technology and the successor to HTML forms.[1] It was designed to tackle the most common problems found in HTML forms (for example, dependency on imperative scripting languages, such as JavaScript) and to ease the authoring of dynamic Web forms.

XQuery: An XML Query Language

XQuery is a powerful 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.[2] XQuery extends XPath, another W3C recommended query language, by introducing additional functionality, such as data grouping and sorting as well as cross-document joins.

References

1. J.M. Boyer, *XForms 1.1*, World Wide Web Consortium (W3C) recommendation, Oct. 2009; www.w3.org/TR/xforms.
2. S. Boag et al., *XQuery 1.0: An XML Query Language (2nd ed.)*, World Wide Web Consortium (W3C) recommendation, Dec. 2010; www.w3.org/TR/xquery.

library, such as Hibernate, to manage data lets an author implement an entire Web application using only Java.

Expanding the Data Tier: Sausalito

Database experts have special needs for designing and developing Web applications. For example, they might need to implement a simple user interface to handle and analyze available data. To use their existing knowledge, they need a development approach based on concepts commonly used on the data tier. The Sausalito XQuery application server (www.28msec.com) provides such a data-centric approach by extending a query language's capabilities to cover the most common client-side and server-side functionality (see Figure 2d).

Sausalito uses declarative XQuery as the base language for developing full-fledged Web applications. In Sausalito, a database expert can author a Web application's presentation tier using declarative XHTML combined with declarative XQuery, instead of having to resort to imperative JavaScript. Sausalito uses XQuery on all three tiers, so the database expert can use the same XQuery code on the server side as well.

When a webpage is requested, the Zorba XQuery processor executes the XQuery server-side code before sending the webpage to the client. On the client side, XQuery in the Browser (XQiB)—which provides browser-related XQuery extensions and includes a JavaScript-based MXQuery XQuery processor—interprets and executes the remaining XQuery code. Sausalito comes bundled with an integrated Web, application, and database server.

From Theory to Practice

We evaluated these three Web application frameworks to obtain a high-level overview and consider their advantages and drawbacks. We focused on eight software product quality characteristics that are part of the ISO/IEC 25010:2011 standard.[9] However, because the standard isn't specifically designed to evaluate Web application frameworks, we included two additional characteristics: time-to-market[10] and support (that is, documentation quality, development community activity, and development tool availability).

Functional Suitability

Sausalito and especially GWT provide rich functionality for developing enterprise-level Web applications with complex business logic. XFormsDB's functionality covers only the most common server-side and database-related tasks, because the framework targets Web designers whose main focus isn't on the application's business logic.

Reliability

GWT, the oldest and most mature of the three frameworks, has been widely adopted around the world, and many public Google products, such as Google AdWords, use it. On the contrary, both XFormsDB and Sausalito were developed as academic projects, though a start-up company was recently established to commercialize Sausalito. The reliability of both frameworks has been tested on numerous Web applications.

Security

XFormsDB provides built-in and easy-to-use security capabilities for authentication and access control. Neither GWT nor Sausalito have such support.

Compatibility

All three frameworks can consume data from third-party systems, such as through Representational State Transfer (REST) APIs. In addition, Sausalito has a built-in solution for building RESTful services, whereas XFormsDB can expose its application data through eXist-db's RESTful API. With GWT, support for RESTful services must be implemented manually with the help of external libraries.

Operability

For an author with little technical knowledge, Sausalito and especially XFormsDB are easy to learn,[11] because they're both based on declarative languages with limited functionality. The adoption rate of both XForms and XQuery, however, is still quite low, despite their being W3C recommendations. In contrast, an author with excellent technical knowledge, especially in Java, might find GWT's learning curve very gentle.

Performance efficiency

Of the three frameworks, Sausalito is the only one designed with scalability in mind; it provides a built-in solution for deploying Web applications to a cloud infrastructure. With GWT, performance isn't an issue, whereas response times in XFormsDB are reasonable but not highly optimized.

Maintainability

Both XFormsDB and Sausalito rely heavily on W3C standards, which increase code modularity and reusability. GWT, alternatively, provides an extensive component library. In addition, both XFormsDB and GWT are open-source projects, so they can be easily modified and extended.

Portability

Development environments for XFormsDB, GWT, and Sausalito all come with executable scripts and a bundled Web server, so they're easy to install and uninstall. Sausalito also provides a built-in solution for deploying production-ready Web applications to a cloud infrastructure.

Time-to-Market

GWT's easier learning curve facilitates the rapid prototyping and development of Web applications. XFormsDB and Sausalito allow rapid prototyping but require additional learning up front.

Support

Both XFormsDB and GWT provide visual development tools, whereas Sausalito provides only development tools for Eclipse. The development community around GWT is very active and its documentation is extensive, whereas the documentation for XFormsDB and Sausalito is less advanced.

Framework Comparisons

According to our evaluation, logic-centric imperative GWT is the most mature and powerful of the three frameworks. It also provides the best support in terms of documentation, activity, and tools. Data-centric declarative Sausalito delivers the most scalable solution with good built-in machine-to-machine communication support. Presentation-centric declarative XFormsDB offers the most attractive solution for authors with little technical knowledge.

However, there are other Web application frameworks representing a tier-centric architectural expansion idea:

- XML-based QAFE (www.qafe.com);
- XQuery-based eXist-db;[8]
- JavaScript-based Helma (www.helma.org); and
- HTML-based Angular.[12]

Figure 3 compares these frameworks according to whether they're imperative or declarative (or a hybrid of the two) and where they originate in the three-tier structure. For each framework, we also indicate the year of origin (the year of its first release or relevant publication). Note that the imperative-versus-declarative comparison is fuzzy—that is, more imperative frameworks appear on the left, more declarative frameworks appear on the right, and the hybrid frameworks are closer to the middle.

We can roughly identify three clusters in Figure 3. The XML- and XQuery-based frameworks originating from presentation and data tiers are typically declarative. In turn, logic-centric GWT and presentation-centric Helma are imperative.

Angular, in the intermediate position, is a hybrid framework.

Although a proper discussion of Web application development approaches is out of this article's scope (a related discussion appears elsewhere[13]), we should mention that model-driven development approaches also exist, such as RUX+WebML,[14] UWE4JSF,[15] and OOH4RIA.[16] In such approaches, the development process involves creating a set of models for an application and then using a declarative graphical language to transform them into a code implementation.

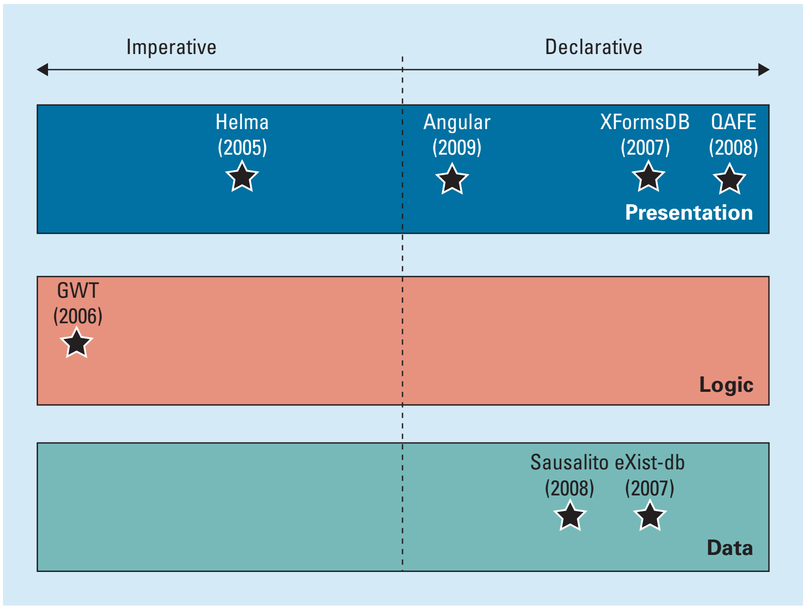

Figure 3. A comparison of the frameworks according to where they appear on an imperative-declarative language scale and where they originate in a three-tier application structure. The XML- and XQuery-based frameworks are typically declarative, logic-centric GWT and presentation-centric Helma are imperative, while Angular is a hybrid framework.

Overall, GWT seems more functional and mature and provided better support. Yet declarative XFormsDB and Sausalito are sufficiently mature to let experts from a corresponding tier author highly interactive Web applications. All of the evaluated approaches offer compelling alternatives to conventional three-tier Web development. With the ongoing trend in deploying Web applications on cloud platforms—where developers have little or no control over the resources the applications are using—the declarative style of Web application development becomes particularly promising. We thus argue that declarative Web application frameworks will play much larger role in the coming years. IT

References

1. G. Alonso et al., *Web Services: Concepts, Architectures and Applications (Data-Centric Systems and Applications)*, Springer, 2003.
2. T. Mikkonen and A. Taivalsaari, *Web Applications—Spaghetti Code for the 21st Century*, tech. report TR-2007-166, Sun Microsystems, June 2007.
3. D. Appleby and J. VandeKopple, *Programming Languages: Paradigm and Practice*, Computer Science Series, McGraw-Hill, 1997.
4. M. Honkala, "Web User Interaction—A Declarative Approach Based on XForms," doctoral dissertation, Dept. Computer Science and Eng., Helsinki Univ. of Technology, 2006.
5. J. Kuuskeri and T. Mikkonen, "Partitioning Web Applications between the Server and the Client," *J. Web Eng.*, vol. 9, no. 3, 2010, pp. 207–226.
6. M. Laine, "XFormsDB—An XForms-Based Framework for Simplifying Web Application Development," master's thesis, Dept. Media Technology, Aalto Univ. School of Science, 2010.
7. M. Honkala, O. Koskimies, and M. Laine, "Connecting XForms to Databases," W3C Workshop on Declarative Models of Distributed Web Applications, 2007; www.w3.org/2007/02/dmdwa-ws/Papers/oskari-koskimies.pdf.
8. W. Meier, "eXist: An Open Source Native XML Database," *Web, Web-Services, and Database Systems*, LNCS 2593, Springer, 2003, pp. 169–183.
9. *ISO/IEC 25010:2011: Systems and Software Engineering—Systems and Software Quality Requirements and Evaluation (SQuaRE)—System and Software Quality Models*, Int'l Standards Organization/Int'l Electrotechnical Commission, 2011.
10. J. Offutt, "Quality Attributes of Web Software Applications," *IEEE Software*, vol. 19, no. 2, 2002, pp. 25–32.
11. M. Åkerberg, "Evaluation of XFormsDB-Based Application Development—A Case Study," master's thesis, Dept. Media Technology, Aalto Univ. School of Science, 2010.
12. M. Hevery and A. Abrons, "Declarative Web-Applications without Server: Demonstration of How a Fully Functional Web-Application can be Built in an Hour with only HTML, CSS & <angular/> JavaScript Library," *Proc. Object-Oriented Programming,*

Systems, Languages & Applications (OOPSLA 09), ACM Press, 2009, pp. 801–802.

13. G. Toffetti et al., "State-of-the-Art and Trends in the Systematic Development of Rich Internet Applications," *J. Web Eng.*, vol. 10, no. 1, 2011, pp. 70–86.
14. M. Linaje, J.C. Preciado, and F. Sanchez-Figueroa, "Engineering Rich Internet Application User Interfaces over Legacy Web Models," *IEEE Internet Computing*, vol. 11, no. 6, 2007, pp. 53–59.
15. C. Kroiss, N. Koch, and A. Knapp, "UWE4JSF: A Model-Driven Generation Approach for Web Applications," *Proc. 9th Int'l Conf. Web Eng.* (ICWE 09), LNCS 5648, Springer, 2009, pp. 493–496.
16. S. Meliá et al., "A Model-Driven Development for GWT-Based Rich Internet Applications with OOH4RIA," *Proc. 8th Int'l Conf. Web Eng.* (ICWE 08), IEEE Press, 2008, pp. 13–23.

Markku Laine *is a doctoral student in the Department of Media Technology at the Aalto University, Finland. His research interests include declarative Web application development and Web performance optimization. Laine received his MSc in communications engineering from the Aalto University. Contact him at markku.laine@aalto.fi.*

Denis Shestakov *is a postdoctoral researcher in the Department of Media Technology at the Aalto University, Finland. His research interests span Web data management and massive data processing, with a focus on scalable Web agents. Shestakov received his PhD in computer science from the University of Turku, Finland. Contact him at denis.shestakov@aalto.fi.*

Evgenia Litvinova *is a research assistant in the Department of Media Technology at the Aalto University, Finland. Her research interests include end-user Web development and Web usability. Litvinova received her MSc in computer science from the University of Eastern Finland. Contact her at evgenia.samochadina@aalto.fi.*

Petri Vuorimaa *is a full professor in the Department of Media Technology at the Aalto University, Finland. His research interests include Web-based services, smart spaces, and mobile media applications. Vuorimaa received his DSc in computer science from the Tampere University of Technology, Finland. Contact him at petri.vuorimaa@aalto.fi.*